Object Discovery with a Mobile Robot

by

Julian Mac Neille Mason

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Ronald Parr, Supervisor

_____
Carlo Tomasi

_____
Jeffrey Forbes

_____
Matthew Reynolds

_____
Ron Alterovitz

Dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in the Department of Computer Science
in the Graduate School of Duke University
2013

Abstract

Object Discovery with a Mobile Robot

by

Julian Mac Neille Mason

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Ronald Parr, Supervisor

_____
Carlo Tomasi

_____
Jeffrey Forbes

_____
Matthew Reynolds

_____
Ron Alterovitz

An abstract of a dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science in the Graduate School of Duke University
2013

Abstract

The world is full of objects: cups, phones, computers, books, and countless other things. For many tasks, robots need to understand that this object is a stapler, that object is a textbook, and this other object is a gallon of milk. The classic approach to this problem is object recognition, which classifies each observation into one of several previously-defined classes. While modern object recognition algorithms perform well, they require extensive supervised training: in a standard benchmark, the training data average more than four hundred images of each object class.

The cost of manually labeling the training data prohibits these techniques from scaling to general environments. Homes and workplaces can contain hundreds of unique objects, and the objects in one environment may not appear in another.

We propose a different approach: object discovery. Rather than rely on manual labeling, we describe unsupervised algorithms that leverage the unique capabilities of a mobile robot to discover the objects (and classes of objects) in an environment. Because our algorithms are unsupervised, they scale gracefully to large, general environments over long periods of time. To validate our results, we collected 67 robotic runs through a large office environment. This dataset, which we have made available to the community, is the largest of its kind.

At each step, we treat the problem as one of robotics, not disembodied computer vision. The scale and quality of our results demonstrate the merit of this perspective, and prove the practicality of long-term large-scale object discovery.

For Ricco, for teaching me the important things.

Contents

List of Tables

List of Figures

List of Algorithms

Acknowledgements

This document is the result of many years of hard work, and that work was not done alone. Far too many people have helped me along the way to list them all, so I will settle for a best attempt. I know that the people I've forgotten will forgive me, and take their thank-you in person.

Thank you to my family, for their relentless confidence in me, and their boundless store of curiosity about what I was up to. Their interest buoyed me when my own was lacking.

Thank you to every teacher I've ever had, and thank you again to a few in particular: Mike Fenster (for pointing out that I could play with robots for fun), Zach Dodds (for pointing out that I could play with robots for a living) and Jeff Forbes (for pointing out that I could play with robots for a living *at Duke*).

Thank you to Ron, for his carefully-considered advice over the many years I was his student. His steady encouragement to take any problem and ask "Why aren't we solving the bigger problem?" led me in many interesting directions, and that perspective is one I will always return to.

Thank you to Bhaskara Marthi, and the extraordinary intellectual community at Willow Garage. Bhaskara's hard work made this document possible, and it contains the fruits of many long conversations and asynchronous multi-time-zone coding ses-

sions. Willow Garage itself put me among many excellent roboticists, and among many excellent robots. Without their support, I would not have been able to pose the question this document poses, nor answer it. The steady hint from Bhaskara (and from the greater Willow Garage community) that the best ideas are those that are running on a robot is another perspective that I gladly take with me.

Thank you to my dear friends Gavin Taylor and Neeti Wagle. As argued by Taylor (2011), we have "shared too many experiences and emotions to trivialize them with a list." Their perspective on the relative merits of pie and work is perhaps the most important that I have gained. Even that does not do them justice. Thanks also to my labmates Christopher Painter-Wakefield and Jason Pazis, for many long conversations and interesting ideas.

Thank you to my committee, for their many interesting observations and the wide variety of perspectives they have brought to my work.

To each and every person listed above, and to the ones I forgot: it has been a privilege.

(Not all of my thanks go to people: I have been funded by Duke University, NSF CAREER award IIS-0546709, NSF IIS-1208245, DARPA HR0011-06-1-0027, and Willow Garage, and without that, there would be no acknowledging to do.)

1

Introduction

When you finish reading this paragraph, put down this document (or your laptop) and look around. When you do, you'll probably see a space full of *objects*: books, computers, dishes, lamps, tables, chairs, staplers, and any number of other things.

Because objects are an important part of your life, they are an important area of study in robotics. A wide variety of classic robotic problems include objects: manipulation ("How do I pick this up?"), tracking ("Where did I leave my car keys?") and change detection ("When did the painting leave the museum?"), among others.

The classic approach to each of these problems begins with a notion of *object identity*: the idea that this object is a "stapler" while that object is a "laundry basket", and this other object is a "coffee cup". The problem of determining object identity from sensor data is traditionally approached with *object recognition*, which has been a field of study for nearly fifty years.[1] Once an object has been recognized, it can be tracked ("Your car keys were seen on the dining room table at 2:53PM"), manipulated (by looking up "laundry basket" in a database of manipulation plans),

[1] As relayed by Szeliski (2010), "Marvin Minsky... asked his undergraduate student Gerald Jay Sussman to 'spend the summer linking a camera to a computer and getting the computer to describe what it saw.' We now know that the problem is slightly harder than that."

or used in other ways.

Object recognition is classically treated as a *supervised* problem, requiring that training data for each object be provided. These data are usually generated by hand, by picking out the object in hundreds of images. In a standard benchmark (the PASCAL challenge of Everingham et al. (2010)), the labeled training data average 416 images of each object class. When this much data is available, modern techniques perform well.

Sadly, applying the supervised-recognition paradigm to the objects around you is problematic. You likely do not have 416 labeled images of your car keys, keyboard, coffee cup, or sunglasses. In fact, relying on supervised recognition makes a key assumption: that *every object a robot could possibly encounter* can be cataloged ahead of time. This assumption might hold in a lab, where only a few objects are relevant, but it fails in general settings: there are simply too many objects!

Consider a robot that operates in your home. Recognizing objects is critical to a wide variety of useful things this robot could do. Now consider the cost of labeling objects: "Robot, bring me my car keys" becomes "Robot, let me show you my car keys in 400 different configurations so that I can request them next time." For some objects, it may be possible to crowdsource training[2] but your keys (among other objects) are unique. Until every home comes equipped with a graduate student to do the labeling, we need something better.

Luckily, we have help. Our hypothetical "home robot" spends its time navigating from place to place, interacting with people, and perhaps manipulating objects or ferrying them from one location to another. All the while, the robot is observing

[2] Another possibility is to solve the problem on the object side: manufacturers could provide downloadable recognition models for every product they make. Another possibility is to embed an RFID tag into every object, a possibility explored by Deyle et al. (2013), among others. Such approaches suffer from something of a chicken-egg problem: manufacturers will do the extra work only when there are enough robots to create demand, but without accurate object understanding, fewer robots will be sold.

its environment, meaning that it will observe the objects in that environment "by accident". These observations will be brief, disorganized, and unlabeled, but over time they will accumulate into many views of many objects. If we can perform *unsupervised* clustering of these views, we won't need to label the objects: the robot can do it for us!

Given the cost of training supervised recognizers, we believe that the unsupervised learning of objects (called *object discovery*) is a requirement to scale robotic systems to general environments. This document describes a way to do exactly that. We focus on the robotics of the problem: rather than operate solely on collections of images, our algorithms leverage the particular benefits of working on-robot, and are robust to the challenges of changing lighting, moving objects, and the fleeting and unreliable nature of our observations.

We factor object discovery into two subproblems: *segmentation* ("Which parts of this image are an object?") and *association* ("Which object is this?"), and present novel approaches to both.

The goal of this document is to prove that object discovery with a mobile robot is not only a good idea in theory, but can be done accurately in practice. To this end, we evaluate our algorithms on two novel datasets: the "Mobile Objects" dataset, (Section 3.2) tuned to objects that move, and the "Willow Garage" dataset (Section 3.3), which focuses on size and generality. The Willow Garage dataset is the largest publicly-available dataset of a robot repeatedly observing an environment.

1.1 Objects

As used above, an "object" is a fuzzy thing. Plates and cups are clearly objects, but what about tables, people, or chairs? To move forward, we need to define "object" more precisely. In common English use, the word "object" is extremely vague, and as Gibson (1986) notes, "The term *object* as used in philosophy and psychology is

so inclusive as to be almost undefinable." Gibson (who is describing a very general terminology for biological perception) proposes two working definitions:

Attached objects

> An attached object is a "substance with a surface that is not wholly closed and is continuous with another surface, usually the ground. It cannot be displaced without breaking the surface."

Detached objects

> A detached object is a "substance with a surface that is topologically closed and is capable of displacement."

In my house, few interesting things are attached objects, although the walls and bookshelves meet the definition. For someone standing outside the building, the *entire building* is an attached object, which is clearly not what we want. Attached objects therefore provide little insight into the objects a robot might want to discover.

Detached objects are closer to what we want. Objects that can be displaced are exactly those objects a robot might want to manipulate, and do not include structural elements like walls. Some objects, like cubicle walls, qualify as "detached", but are likely of little interest to a robot. (Cubicle walls are best treated as walls, even if they could be moved with sufficient effort.)

Supervised object recognition implicitly provides a third definition of "object": an object is anything we care enough about to train a recognizer for. Unlike Gibson's definitions, this is relentlessly practical, as each class of objects is defined explicitly. As noted above, we pay for this concreteness up front, when we train our recognizers.

As Gibson's definitions are too vague to be implemented directly, and training recognizers for every object is impractical, we necessarily stand somewhere in between, and need a heuristic definition. Starting with detachable objects, we consider some desiderata:

- We are interested only in *indoor objects*, as our robot (described in Section 3.1) operates only indoors.

- We are most interested in *manipulable* objects. Not all objects are manipulable (and not all manipulable objects are manipulable by our robot), but we believe that objects that our robot could pick up if it so chose are of particular value.

- Finally, we limit our attention to objects that can be *discovered passively*. In keeping with the common case in robotic mapping, we want our robot to be able to discover objects without actively manipulating the environment. Active object search presents interesting directions for future research (Section 7.2), but is not the focus of this work.

With all of this in mind, we define two heuristic forms of "object", which provide the basis for the segmentation algorithms detailed in Chapter 5.

Mobile Objects

A *mobile object* is a detached object that is actually seen to move. Here, "move" does not mean that we see the object change position "live", as in a video sequence. Instead, we assume that the world is "semi-static": each observation is of a static world, but objects can move (or, most likey, be moved by some other agent) while the robot isn't looking. This is motivated by our interest in passive discovery: we can't manipulate objects directly, nor have a person move them for us on command. See Section 5.2 for details.

Supported Objects

A *supported object* is a detached object that is resting on a (horizontal) surface. In particular, our techniques rely on the observation that "resting on something flat" is a very common state for objects. We discard the floor, as it trivially supports everything. See Section 5.3 for details.

1.2 Terminology

For clarity, we define some terminology here that is used throughout the document.

Frame

Our robot (Section 3.1) has an attached color and depth (RGB-D) camera. Our robot is also self-localized, meaning that it knows its own location and orientation. We call an RGB-D image pair, combined with the localization estimate of the robot's position at the time the pair was captured, a *frame*.

Segment

A *segment* is a connected subset of the pixels in a frame. As we are discovering objects, the segments we care about are those whose pixels correspond to an object. Frames can contain zero or more objects, so there can be zero or more (disjoint) segments extracted from a single frame. Segmentation (the subject of Chapter 5) is the process of computing segments from frames.

Instance

An object *instance* is a particular object in a particular location. See Figure 1.2 and Figure 1.3 for examples. Should the object move, the object in the new location creates a new instance.

Class

An object *class* is a particular type of object, independent of location. Every segment in Figure 1.2 is in the class "biped", despite being in a variety of locations. Similarly, every segment in Figure 1.3 is in the class "houseplant."

Run

A *run* is a single robotic data-collection event. This can be thought of as "boot the robot up, collect some data, and then shut the robot down." (In practice,

the robot is almost never turned off.) Our robot and datasets are detailed in Chapter 3.

1.3 Object Discovery

We factor object discovery into two subproblems:

Object segmentation

Segmentation seeks to determine which parts of the world correspond to objects. On a frame-by-frame basis, segmentation consists of computing connected groups of pixels (segments) thought to be "on an object". As there can be many objects in a single frame, there can be many segments computed from a single frame. See Chapter 5 for a detailed discussion of the segmentation problem and our approaches to it.

Object association

Association seeks to group these segments into clusters that correspond to instances or classes of objects. Unlike recognition (which also groups observations into classes), association does not provide meaningful labels: clusters are "object 1" or "object 6", not "coffee cup" or "laundry basket". Nevertheless, "this object is the same as that object" is a powerful label to have. See Chapter 6 for a detailed discussion of association and our approaches to it.

1.4 Data Path

This document describes the interconnected pieces of a complete system. A high-level overview of the path data take through the system can be seen in Figure 1.1. Chapter 4 includes a detailed discussion of our software layer.

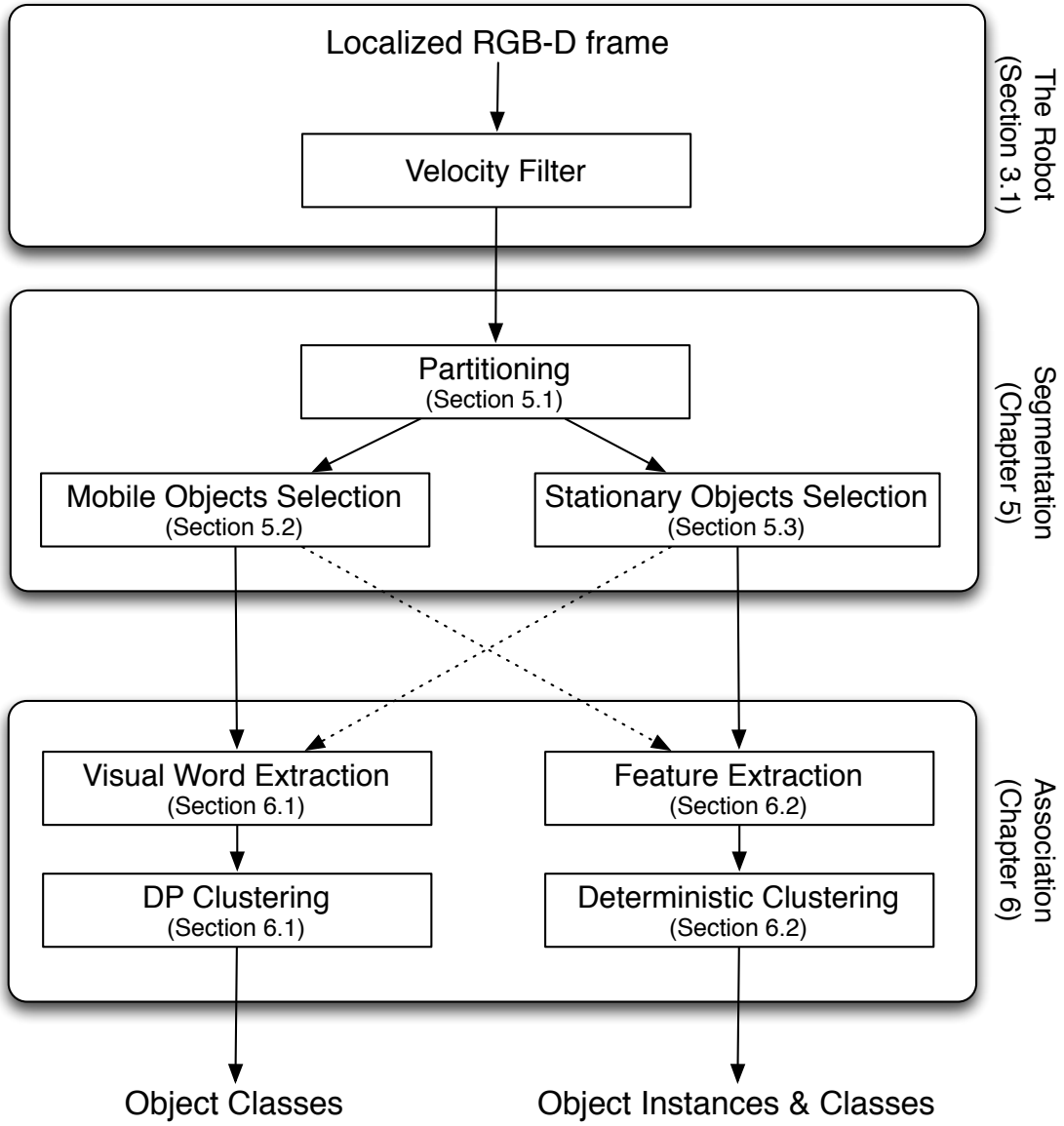
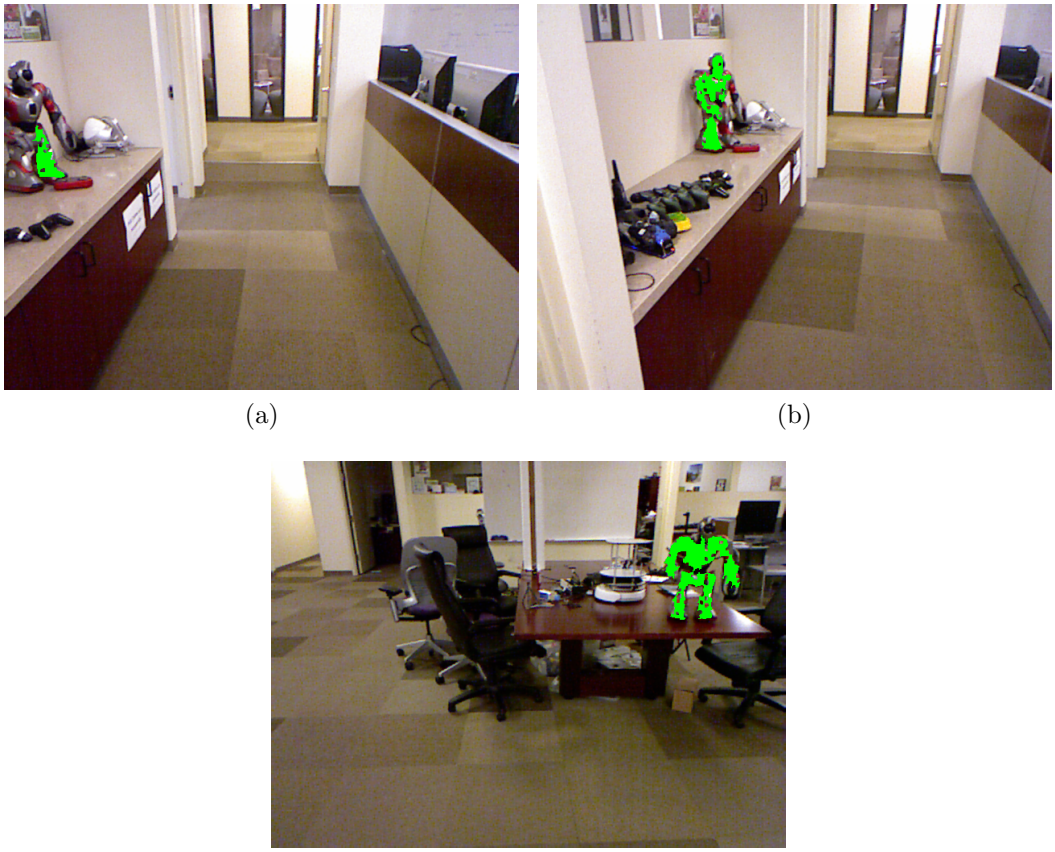


FIGURE 1.1: The data path. Boxes with square corners denote algorithmic steps, and boxes with rounded corners correspond to our factorization of object discovery. Solid arrows correspond to data paths actually taken in this work; dotted arrows correspond to paths that could be taken, but were not used in our experiments.



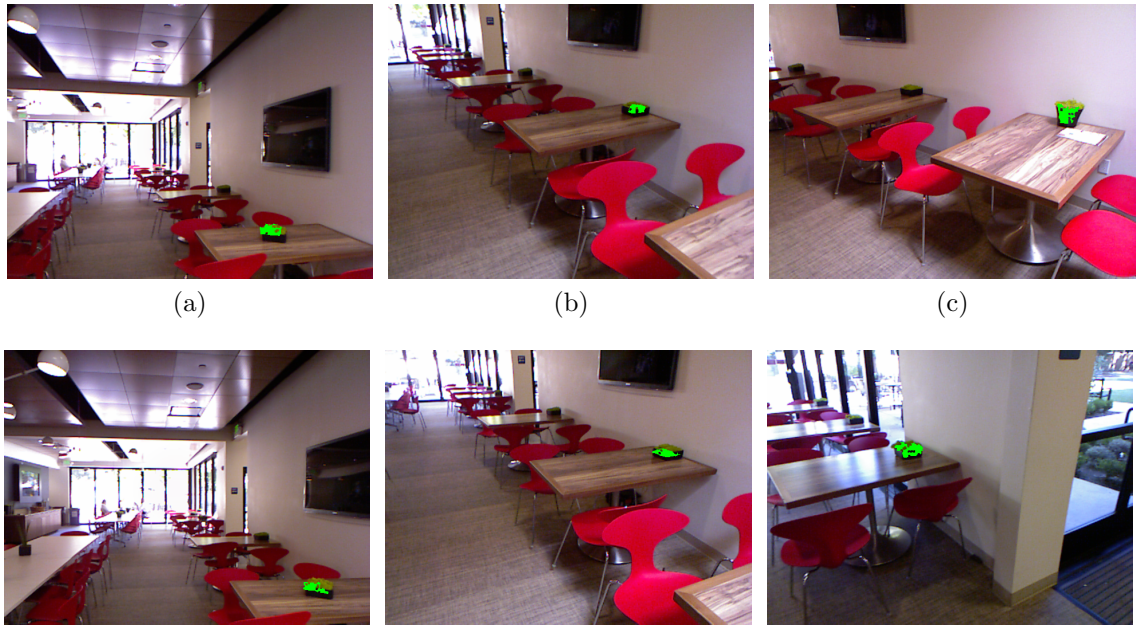
(a) (b)

(c)

FIGURE 1.2: The distinction between "instances" and "classes", which are defined in Section 1.2. The segments in (a) and (b) are both of a bipedal robot in the same physical location; as a result, they both belong to the same object instance. By contrast, because the robot has moved, the segment in (c) belongs to a different instance. As all three segments are of the same object, they belong to the same class (in our manual labeling, "biped"). *This figure is best viewed in color.*

1.5 Chronology

This document is the result of continuous thread of work that has resulted in two published papers and one paper currently under review. The first paper (Mason and Marthi, 2012) focused on mapping the locations of the objects in an environment (called *semantic mapping* in the robotics literature), but under weak perceptual assumptions. In particular, it stood apart from prior work in semantic mapping by assuming that object recognition was not available. Lacking recognition limited



(a) (b) (c)

(d) (e) (f)

FIGURE 1.3: Several segments belonging to the same class, as defined in Section 1.2. First column: two segments corresponding to the same instance. Second column: two segments corresponding to the same object class as the first column, but a different instance (because the object has moved). Third column: two other segments belonging to the class "houseplant." *This figure is best viewed in color.*

the types of semantic inference that can be done; however, the work demonstrated prototype implementations of both *semantic querying* and *change detection*, which are discussed in Section 4.3. Requiring only weak perceptual assumptions allowed this approach to scale to the Willow Garage dataset (which was collected for, and published in, that paper).

Following the first paper, we turned our attention to the association problem, again without supervised recognition. Drawing some inspiration from the recognition literature, our second paper (Mason et al., 2012) addressed class-level association by focusing on object appearance. A key problem in the unsupervised setting is that the number of true clusters is unknown; our solution was a probabilistic formulation of object appearance based on a Dirichlet process. In parallel, the second paper investigated the use of mobile objects to improve segmentation.

The second paper demonstrated high performance, but on limited inputs: it requires that objects be mobile to be discovered, and the probabilistic formulation of appearance requires fairly close-range views of textured objects.

In the work currently under review (and described in detail in this document), we closed the loop, achieving accurate association in fully general data. Rather than focus solely on appearance, our newest approach combines appearance, size, shape, and object location to perform association, and does not require object motion for segmentation. Like our first work, it scales to the Willow Garage dataset; unlike the first work, it accurately associates objects across time and space.

This document is laid out by topic, not in chronological order. In Chapter 2, we situate our work in the robotics literature and review relevant background material. In Chapter 3, we describe in detail the robot we used to collect our data, and the datasets on which we evaluate our algorithms. In Chapter 4, we describe the software infrastructure that supports our goals of large-scale, long-term object discovery. Chapter 4 also describes the prototype semantic-querying and change detection applications from our first paper. In Chapter 5, we describe and evaluate our segmentation algorithms, and in Chapter 6 we do the same for association. Finally, Chapter 7 draws conclusions and discusses directions for future work.

2

Background

Object discovery sits in the intersection of many classic problems in robotics and computer vision. Here, we give a brief overview of related problems and their overlap. Surprisingly, robotic object discovery has seen very little attention; our nearest neighbors (discussed at the end of this chapter) focus on very small scenes, or treat the problem as one of disembodied computer vision.

2.1 Maps

The construction and use of maps is fundamental in mobile robotics. Robots need maps to plan actions, explore in an intelligent fashion, and report their findings back to a human user. Maps overlap and interact with the object discovery problem in several ways. First, they provide the ability to *localize* the robot: that is, to accurately determine its position in a fixed coordinate frame. Given a localized robot, we can (and do) use the metric location of segments as an input to our discovery algorithm. Second, localization supports planned navigation: although a robot could gather views of objects through a random walk, smarter planning requires a map. (Autonomous navigation is the backbone of our data-collection

regime: see Chapter 3.) Finally, maps can include object locations. In Chapter 1, we give the example of asking the robot "Where did I leave my car keys?" To answer this question, the robot needs a map: either a metric representation ("Your keys are at coordinates (x, y, z)") or a higher-level semantic representation of location ("Your keys are in the kitchen."). Such semantic representations are naturally built atop metric maps; see Section 2.4.

This multitude of uses has made robotic mapping a field of study for nearly thirty years, going back to (at least) the work of Moravec and Elfes (1985). That work introduced the *occupancy grid*, the original mapping primitive. In an occupancy grid, the environment is discretized into a uniform grid at some fixed resolution. Each resulting grid cell is then given an *occupancy*, which encodes the presence or absence of an obstacle. This representation is intuitive, and particularly easy to visualize in two dimensions: see Figure 2.1.

Occupancy grid maps are not limited to two dimensions. In work by Fairfield (2009), and in unpublished work of mine (described in my preliminary exam document (Mason, 2010)), occupancy grids are extended to three dimensions. As this leads to an explosion in the amount of information needed to store the map, both my work and Fairfield's use *octrees* (Meagher (1982)) to compress uniform spatial regions.

Occupancy grids have also been extended to include more than just occupancy information. Moravec (2002) describes a small, dense three-dimensional occupancy grid to include color information from a stationary three-camera sensor rig. More recently, in Mason et al. (2011) we added RGB color to a large octree-based three-dimensional occupancy grid, and used that information to localize a mobile robot using only a single camera. The OctoMap library by Hornung et al. (2013) is an open-source implementation of 3D octree maps with attached extra information.

FIGURE 2.1: A two-dimensional occupancy grid map. This map is of the interior of Willow Garage (a general office environment, and the site of our experiments) and was built using the GMapping algorithm aboard a PR2. This map is a horizontal slice through the environment, taken roughly 30 cm above the floor. In this map, the colors range from black (high occupancy) to white (low occupancy), and "unknown" is represented by gray. At full resolution this map is 2425×1900 pixels. Each pixel represents a region 2.5 cm on a side. The three thick black lines were added by hand to keep the robot from planning paths through areas that had become impassable after the map was built.

2.2 Localization

Our work in object discovery leans heavily on the ability to localize the robot in its environment. For completeness, we discuss localization here.

Consider a robot. It has a *pose*, which we denote by $\mathbf{p}$. In our case, this pose has three dimensions: coordinates x and y (in meters) and an orientation θ, in radians. Our goal is to accurately track $\mathbf{p}$ as the robot moves ("robot tracking") and we may also wish to determine to determine the robot's position from scratch (the "kidnapped robot" problem).

Modern (that is, probabilistic) localization algorithms operate by maintaining a *belief state* over the robot's pose, in the form of a probability distribution. The details of how the belief state is represented, and how it is updated in response to robot actions and sensor readings, define the localization algorithm.

In their seminal paper, Dellaert et al. (1999) proposed an algorithm called *Monte Carlo Localization*, which is an application of the general technique of *particle filtering* to localization. Particle filtering is not only the state of the art in localization, but particle filters form the basis of the simultaneous localization and mapping (SLAM) algorithms we discuss in Section 2.3.

Particle filters represent the belief state nonparametrically, using n samples ("particles"), each of which is a deterministic hypothesis about the robot's pose. As such, each particle stores a pose $\mathbf{p}$, and a *particle weight* w, which encodes the confidence that this particle is the true $\mathbf{p}$. These weights are normalized to sum to 1, meaning that the complete set of particles defines a discrete distribution over poses.

At each timestep, the robot performs an action and takes a sensor reading. The particle filter updates the particle poses by drawing new poses from a *motion model* based on the action; the particles are then reweighted by combining the known map with a *sensor model* and the sensor reading. Finally, particles are *resampled*,

using the discrete distribution to duplicate high-probability particles and delete low-probability particles.

As each particle must be updated at each timestep, the cost of maintaining the filter is $O(n)$. Choosing n manually is difficult, as it must trade off between representational accuracy and computational cost. The solution is the *Adaptive* Monte Carlo Localization algorithm (AMCL) by Fox (2003), which examines the belief state to determine the necessary number of particles.

Our robot (Section 3.1) uses AMCL (as implemented in ROS; see Section 4.1.1) for localization. As we do not have ground-truth poses against which to compare, we cannot directly measure the accuracy of our localization estimate. However, experience suggests that our errors are at most a few centimeters.

2.3 SLAM

As described above, localization assumes a static map. The harder problem is to localize the robot in an unknown environment: this is often done by building the map on the fly, which is called *simultaneous localization and mapping*, or SLAM. While we are not performing SLAM in this work, our static map has to come from somewhere, so we briefly discuss SLAM for completeness.

The fundamental difference between SLAM and localization is that the robot must build a map, even though the robot *does not know (exactly) where it is.* The key insight of successful SLAM algorithms is that SLAM, like localization, is a state tracking problem, and should be handled probabilistically. In localization, the state is the robot's pose; in SLAM, it is the robot's pose *and map.* This greatly increases the dimensionality of the problem; localization tracks a three- or six-dimensional state, while SLAM tracks a state that scales with the size of the world.

The "localization" step in SLAM must proceed in an incomplete map, which is a source of potential error. Over time, these errors will accumulate; when the

robot revisits a location ("closes a loop"), the accumulated error will keep the map from lining up correctly. To succeed, SLAM algorithms must represent uncertainty about the map until a loop closure; as representing maps is expensive, this is not straightforward. The DP-SLAM algorithm (described in Eliazar and Parr (2003, 2004, 2005) and Eliazar (2005)) uses a copy-on-write strategy to make maintaining many maps as cheap as possible, while the GMapping algorithm (described in Grisetti et al. (2005, 2007a,b) and Stachniss et al. (2005)) relies on scan matching to minimize localization errors, and a delayed-resampling scheme to "forget" map uncertainty as rarely as possible.

For a more detailed discussion of these, and other, SLAM techniques, we direct the reader to Mason (2010) and Thrun et al. (2005).

2.4 Semantic Mapping

As described above, localization and mapping are *metric*: positions are measured in meters and radians in some fixed, shared coordinate frame. In contrast, the study of *semantic mapping* focuses on maps that include higher-level constructs like objects, rooms, and available actions (often embedded in a metric map). Pronobis (2011) defines semantic mapping as "creating a representation of the environment which grounds human spatial concepts to instances of spatial entities", and uses examples like "hallway" and "room", which are distinct from the purely metric form of an occupancy grid.

As Pronobis uses this definition, objects (and, by extension, their properties) are also "spatial concepts". Semantic mapping, then, is the assignment of human-readable ("semantic") labels to percepts: we call this *semantic labeling*. Semantic labels can be inferred from the data ("The consensus color of this segment is green.") or provided by a human ("This segment is a houseplant.") From this point of view, object recognition is a semantic labeling algorithm: it applies labels (object identi-

ties) to segments.

Semantic labels can take many forms. Nüchter et al. (2006) present a good example of a very simple semantic label. Their work builds three-dimensional point clouds of indoor environments. Each point is then labeled as "floor", "ceiling", or "object" based on a local geometric analysis. Sadly, their "object" label appears to mostly occur on walls, which are not objects by our definitions. Although this ternary labeling is extremely low-level, one can imagine using it in a variety of ways. For example, "floor" points could be subtracted from the set of possible obstacles, thereby implementing a form of ground-plane subtraction.

A higher-level type of semantic label is topological place-labeling. As an example, Galindo et al. (2005) build an occupancy grid map of their robot's environment, and then segment that map into several rooms, each of which is a "large open space" connected by narrow openings. These rooms are then assigned types ("kitchen" or "living room") based on the objects contained within. In Mason and Marthi (2012), we took a simpler approach: we observed that there were only five semantically distinct locations in our environment, and labeled each "room" manually, which took only a few minutes. These place labels support giving the robot commands like "Drive to the living room" or reporting facts like "Your car keys were in the living room seventeen minutes ago".

Objects are by far the most interesting component of semantic maps. Labeled objects with metric poses enable very straightforward commands: the vague "Go fetch my car keys" becomes the well-defined "Navigate to the location of my car keys, invoke the car-keys grasping routine, and then navigate back." Labeled objects also enable other types of analysis. For example, work by Galindo et al. (2005, 2008) and Ranganathan and Dellaert (2007) uses object labels to automate labeling places. They exploit the tendency of objects to appear only in certain types of rooms. For example, Galindo et al. (2005) define a "kitchen" to be a room with at least one stove,

at least one coffee machine, and at most zero bathtubs, among other constraints.

General object recognition remains an open problem, leaving complete semantic labeling equally open. Supervised object recognition techniques suffer from a need for high-resolution data and an accurate database of objects to be recognized. In the context of semantic mapping, Rusu et al. (2009a) use three-dimensional object shape to match objects (segmented at close range) to a small set of known possibilites. In constrast, Vasudevan et al. (2007), Zender et al. (2008), and Blodow et al. (2010) provide examples of *feature-based* techniques, in which feature points are extracted from data. Vasudevan et al. (2007) and Zender et al. (2008) use the appearance-based Scale Invariant Feature Transform (SIFT) feature introduced by Lowe (1999). Blodow et al. (2010) use the Fast Point Feature Histogram (FPFH) and Global FPFH (GFPFH) features, introduced by Rusu et al. (2009c) and Rusu et al. (2009b) respectively. (G)FPFH features operate on point clouds, matching object shape, not appearance. Each of these techniques are validated only on close-range, high-quality data with small object databases.

These techniques make strong perceptual assumptions, which limit what a robot can do while using them. In our experience (and in our data) objects are seen from varying (and long) range, from multiple points of view, and with unpredictable occlusion and lighting. Furthermore, as we are doing object discovery, we lack exhaustive training data: these unreliable observations are all we have.

Object discovery provides a very specific, and very weak, form of semantic label. Rather than labels like "houseplant" or "car keys", discovery can only provide "is-the-same-as" labels: "These segments are all in class 1". Such labels are hardly useful in plan specification or manipulation planning. However, if the labels are accurate, there exists a mapping between these arbitrary labels and human-readable class labels. For example, class 1 might be "car keys" and class 5 "coffee cup." As I learned when labeling our data for analysis, labeling *clusters* of segments ("This

cluster of segments is all coffee cups") is far easier and less error-prone than labeling segments one at a time.

2.5 Cosegmentation

Cosegmentation, as introduced by Rother et al. (2006) (and expanded upon by Vicente et al. (2010)) is an interesting variation on the object discovery problem. Cosegmentation considers the case of jointly segmenting the *same object* from two images at once. The authors note that image segmentation is, in general, extremely difficult. Their technique leverages an interesting observation: suppose that you have two images which are known to contain the same object (but the object has not been segmented). By looking for the part held in common by both images, it should be possible to more-accurately segment the shared region.

Although the goals of cosegmentation are similar to those of object discovery, they are key differences. Cosegmentation assumes only two images, and assumes that the *same object* appears in both images. We define object discovery as segmentation followed by association. Cosegmentation uses an existing association (the two images) to improve segmentation; in this sense, cosegmentation is object discovery "in reverse". As will be seen in Chapter 6, association, not segmentation, is the more challenging subproblem.

2.6 Object Discovery

Our discussion to this point has been of work that overlaps with our own. However, there is also prior work in object discovery itself. This work can be broadly broken into two categories: *live motion* and *scene differencing*. Live motion approaches observe the world changing moment-to-moment, and use this motion to segment and associate objects. By contrast, scene differencing approaches construct one or more static maps of an environment, allowing some time to elapse between maps.

Changes are then detected by comparing these maps, and these changes are grouped into objects.

2.6.1 Live Motion

Motion has been a topic of constant inquiry since the very earliest days of computer vision, and a complete summary of motion-related work is beyond the scope of this document. However, we note some examples here. The factorization method of Tomasi and Kanade (1992) operates on a set of tracked image points, factoring them into matrices that represent the shape and motion of a single (rigid) object. Costeira and Kanade (1998) extend this model to the case of tracking multiple objects undergoing independent motion. When motion is not independent or tracks are of particularly poor quality, techniques like that of Rao et al. (2008) can be applied.

In robotics, Sanders et al. (2002) present an early example of live-motion object discovery. Their technique uses several fixed cameras, and analyzes the time-series of each image pixel to group pixels according to how they change. Because it operates in pixel space, this technique is limited to fixed cameras. Modayil and Kuipers (2004) use a two-dimensional occupancy map to classify sensor readings as "static" or "dynamic"; dynamic readings are then clustered and tracked. Southey and Little (2006) combine stereo vision and optical flow techniques to segment moving objects in video, and then use visual features to group these segments. Ayvaci and Soatto (2012) use motion in video to find occlusion cues, which are then integrated to partition the image into depth layers; discontinuities in these layers correspond to object boundaries. Finally, Sivic and Zisserman (2006) do frame-to-frame tracking in video, and aggregate groups of points that move together to segment objects. The tracked frames become exemplars of the object's appearance, allowing object-level queries.

Finally, should the robot be allowed to manipulate its environment directly, ac-

tions can be chosen that allow efficient disambiguation of objects. Krainin et al. (2011) and Bersch et al. (2012) provide examples of using manipulation to support object discovery and modeling.

2.6.2 Scene Differencing

Biswas et al. (2002) present an early example of scene differencing, using two-dimensional occupancy grids. Their maps are built using sonar, and an expectation maximization (EM) procedure is used to infer object models. Anguelov et al. (2002) extend this technique to include object class templates. A key property of these techniques is that they assume that the world is static in the short term: each map is built assuming a static world, and the changes between maps provide the objects. Kang et al. (2009) present an interesting variation that operates solely on images. A single image is explained as a composite of parts of images from a corpus. This can be thought of as learning a model for the background, and detecting the things that don't match it.

Herbst et al. (2011b) present an object discovery technique based on taking the difference between two three-dimensional maps. Unlike the grid-based maps described in Section 2.1, their map is based on *surfels* (Pfister et al. (2000)). A surfel is a "surface element", analogous to a "picture element" (that is, a pixel). A surfel is a small surface patch, posed in six degrees of freedom, with an associated color. Using the SLAM algorithm presented by Henry et al. (2010), they build two maps: the first with a set of objects, and the second after moving some of the objects. This is what we call a "mobile object" in practice: this technique discovers objects that moved in the interval between the maps. These maps are then registered, and a probabilistic model of their sensor (an RGB-D camera) is used to determine which surfels have changed. Those that have changed are grouped into objects.

Herbst et al. (2011a) expand upon this technique to more than two input maps.

To work, both techniques require the precise alignment of very high-quality, dense maps. While map registration can be done automatically, it limits the applicability of these techniques to fairly small environments; their experiments are done on tabletop-sized examples. Even assuming accurate large-scale multi-map registration, the computational cost of their technique scales with the observed surface area, as every surfel must be examined (and stored). When these requirements can be satisfied, these techniques demonstrate excellent results.

Finally, Kang et al. (2011) approach the object discovery problem in a general dataset of images from "daily living"; high-resolution, close-range images of a variety of objects in an indoor setting. They combine a hierarchical oversegmentation with visual features and color information to perform unsupervised clustering. In the nearest neighbor to our work, Kang et al. (2012) address the sparsity of the object views in the "daily living" dataset, extending their earlier work to leverage a database of product images scraped from the internet.

By contrast, we collect views using a robot. We consider our work and Kang's to be complementary; object views collected in the local environment could be augmented with product images, likely improving performance. However, many of the objects our robot encounters are truly unique to the environment: they are unlikely to appear in any product database. Furthermore, our approach allows *specificity*: our association algorithms need only distinguish among those objects that actually appear in the environment, not those from the larger set. This eases the association problem.

While Kang et al. (2011) and Kang et al. (2012) provide data, they do not provide an implementation. As my techniques require robotic data, I cannot evaluate my algorithms on their data. As they do not provide an implementation, I cannot evaluate their techniques on my data. As a result, no quantitative comparison is possible.

3

Datasets

This document is about object discovery with a mobile robot. This obligates us to deploy our algorithms on a real robot, and to analyze our performance on real robot data. To this end, we collected two datasets. The "Mobile Objects" dataset (discussed in Section 3.2) was designed to study a particular model of object movement, and consists of three runs of increasing complexity. The second dataset ("Willow Garage", Section 3.3) was designed to be both very large and very general. The Willow Garage dataset is an extensive exploration of an office environment, repeated roughly three times a day, for six weeks (totalling 67 runs). As such, it is by far the largest dataset of its kind. No effort was made to simplify the data: they include people, other robots, navigation and localization errors, and a wide variety of objects, which appear in a wide variety of settings and lighting conditions.

As the first dataset of its kind, the Willow Garage dataset is also the first benchmark for robotic object discovery.[1] By performing well on it, we make our case for robotic object discovery; by making it available to the community, we make it

[1] Because it includes repeated observations of the same building, it would also be a natural benchmark for a variety of other perception problems, including SLAM, and for the future work we discuss in Chapter 7.

possible for future work to compare against our results.

We begin with a discussion of the robot used to collect our data.

3.1 The Robot

With one exception (noted later), all of our data were captured with a Willow Garage "Personal Robot 2" (or PR2), examples of which can be seen in Figure 3.1 and Figure 3.2. The PR2 is a large, mobile, two-armed, general-purpose robot, intended for research in perception, planning, and manipulation, and built for indoor environments.

The PR2 rides on four power casters, allowing holonomic motion.[2] The PR2 also has two arms with attached grippers, a mobile "spine" that allows it to grow taller or shorter, and a "head" with controllable yaw and pitch.

The PR2 sensor suite is extensive. Mounted on the base (roughly 30 cm above the floor) is a Hokuyo UTM-30LX laser rangefinder, which is used for localization (Section 2.2) in our applications, and was used (not by us) to build the map shown in Figure 2.1. Each arm contains an RGB camera for close-range investigation of objects during manipulation. Above the arms is another UTM-30LX, (the "chest laser") attached to a tilting mount. The UTM-30LX scans in a plane; via repeated pitching, the chest laser collects three-dimensional laser data. These three-dimensional points are used for (among other things) obstacle avoidance during autonomous navigation, as described by Marder-Eppstein et al. (2010). The PR2 head carries five cameras, (two stereo pairs with different baselines and one high-resolution camera), and a texture-projection system (as described by Konolige (2010)) for improved stereo vision performance.

The release of the Microsoft Kinect effectively made the stereo system obsolete.

[2] Strictly speaking, it allows *quasi-holonomic* motion, as it may be necessary to reorient the casters before the robot moves. This distinction has little practical effect.

The Kinect provides high-resolution (640×480) depth data at 30 Hz. As quantified by Khoshelham and Elberink (2012), the Kinect is accurate to roughly 2.5 cm when measuring distance of 3 m. This accuracy, combined with the high resolution and framerate, has made the Kinect the standard depth sensor for indoor mobile robotics. The Kinect also has an RGB camera. At 30 Hz, the Kinect can capture both RGB and depth at 640×480 pixels; in high-resolution mode, the RGB image is captured at 1280×1024, and the depth image at 640×480, but at only 15 Hz. Note that 1280 is twice 640, but 1024 is not twice 480; in the vertical direction, the RGB camera has a larger field of view than the depth camera. The solution is a simple truncation: the topmost 960 rows of the RGB image correspond to the pixels in the depth image. (In particular, high-resolution RGB pixel (i, j) corresponds to pixel $(i/2, j/2)$ in the depth image.) This truncation is apparently done automatically in low-resolution mode: the RGB and depth pixels correspond one-to-one in that case.

Sadly, the RGB and depth cameras do not have synchronized shutters. When the camera is moving, some frames will contain mis-synchronized RGB-D pairs, which show a noticeable offset. Our data-collection regime includes a velocity filter, which discards frames taken during rapid robot (or head) motion. While this removes many such synchronization errors, it does not eliminate them. An example error can be seen in Figure 5.2e.

A single data collection run involved a PR2 operating in Willow Garage, recording data. Every run used the Kinect as the primary sensor, and relied on the base laser for localization. Those runs that relied on autonomous navigation used the full navigation system (the "nav stack") described by Marder-Eppstein et al. (2010). Those that were teleoperated used the PR2's standard teleoperation system: a bluetooth-connected video game controller. Importantly, our techniques do not require the full capabilities of the robot: in fact, we require only a Kinect, mounted fairly high up (ours was roughly 1.4 m off the floor), on a localized mobile base. Autonomous data

FIGURE 3.1: A PR2 observing a scene. Visible here are the two-dimensional laser user for localization (centered on the base, just above the vent opening), the arms (folded in front of the robot), the three-dimensional chest laser (just to the left of the "PR2" label) and the head, with attached Kinect (pointed at the table). For a detail of the head, see Figure 3.2.

gathering of course requires access to autonomous navigation.

3.2 The Mobile Objects Dataset

In Chapter 1, we defined a *mobile object*: a "detached object" (according to Gibson's definition) that is actually seen to move. Hypothetically, this motion is a strong sign of "objectness", and should give us leverage in segmenting mobile objects. To test this hypothesis, we collected the Mobile Objects dataset. The pattern for each run is this: several objects are placed in the environment, the robot is teleoperated into position to observe them for a few moments, and then teleoperated away. At some later point, the objects are removed, and the robot is teleoperated to re-observe the location for a few moments. Note that our goal is not to determine how mobile objects are "in the wild", but to investigate mobility's value for segmentation. As it includes good views of several well-textured mobile objects, this dataset could also

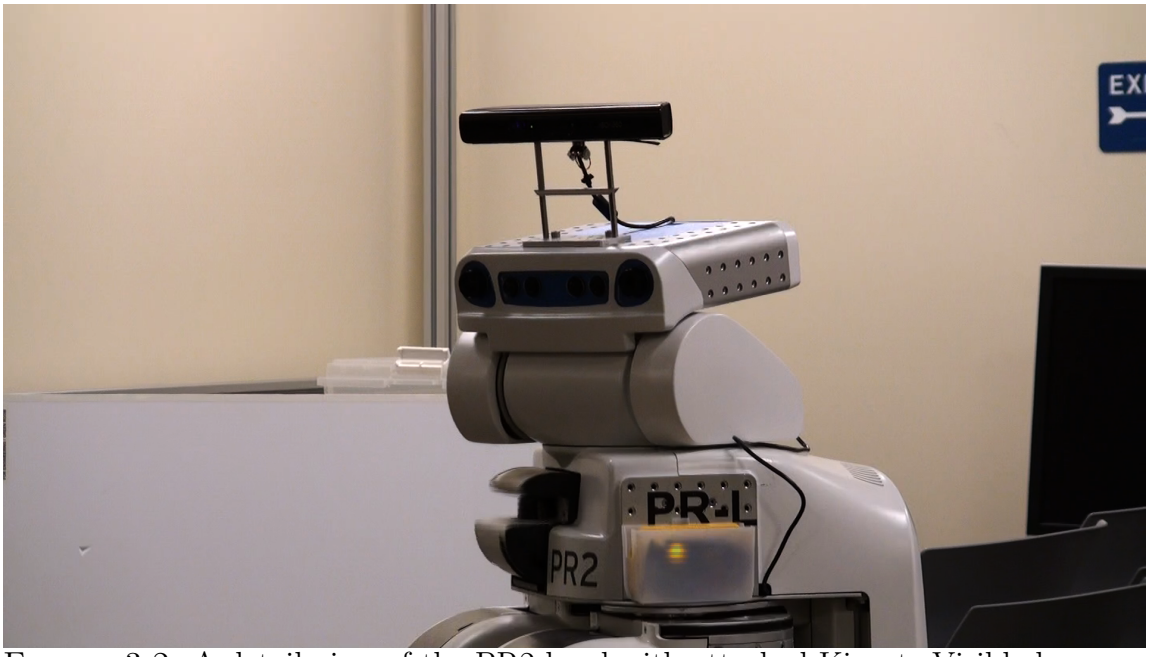


FIGURE 3.2: A detail view of the PR2 head with attached Kinect. Visible here are the chest laser (facing to the left of the picture), Kinect (atop the head) and head sensor package (the "eyes"). The glowing light below the PR-L badge (this robot is named PR-L) is the wireless runstop.

be used as a benchmark for object recognition, and in particular the unsupervised training we discuss in Chapter 7.

The mobile objects dataset consists of three runs, which we call:

Small

The small run is a fixed-camera example, collected as a sanity check. The run consists of 101 frames of a static, empty scene, followed by 135 frames in which two objects have been added, and then 114 frames in which the objects have been removed. Example images can be seen in Figure 3.3. The hand-segmentation results in 270 segments, and two unique objects. Unlike every other run in either dataset, this run was done without a robot. Both the RGB and depth images in this run were recorded at a resolution of 640×480 pixels.

Medium

The medium run was collected on a PR2, navigating in Willow Garage. The robot observes a table (with objects), looks away while the objects are removed, and observes the table again. The robot then travels roughly 18 m, and repeats this process with a countertop that contains the same objects. The same objects were used in both places to test data association across locations, with the commensurate changes in lighting, perspective, and so on. There are four unique objects, 394 segments found by hand, and 484 total frames. Example images can be seen in Figure 3.4. This run recorded RGB at 1280×960 pixels, and depth at 640×480 pixels.

Large

Like the medium run, the large run was collected on a PR2 navigating in Willow Garage, but over a total distance of 181.5 m. There are two passes in this run. In the first pass, the robot observes several objects in each of four locations. In the second pass, all the objects are removed, and the locations are re-observed. There are a total of seven unique objects, 419 segments found by hand, and 365 frames that pass the velocity filter. The frame count is lower than the medium dataset because the robot was not allowed to linger as long in any location. Example images can be seen in Figure 3.5. As with the medium run, RGB is recorded at 1280×960 pixels and depth at 640×480 pixels.

The Mobile Objects dataset, including the ground-truth segmentation and labeling, is publicly available for download at `http://ros.org/wiki/Papers/IROS2012_Mason_Marthi_Parr`.

(a) An image from before the objects are added to the scene.

(b) An image where the objects are visible.

(c) An image after the objects are removed.

FIGURE 3.3: The three phases of the small run from the Mobile Objects dataset (Section 3.2). The abrupt change in illumination from (a) is the Kinect setting the exposure automatically, which cannot be turned off.

3.3 The Willow Garage Dataset

Our second dataset is much larger, and serves a very different purpose. Rather than guaranteeing object mobility, the Willow Garage dataset seeks to be *large, general* and *realistic*. Consider the "home robot" described in Chapter 1. Such a robot lives in a general setting, and interacts with both the world (objects, walls, and so on) and other agents (people and other robots). The size and complexity of such environments lead to several unique challenges, and any successful robotic object discovery algorithm must deal with them gracefully. Therefore, to reasonably validate our approach, we must devise a dataset that includes them:

Size General environments are large and varied: they have many hallways, doors, rooms, objects, and obstacles. Our dataset must therefore be large and feature a mixture of rooms, common areas, and hallways.

Duration

To collect a meaningful number of object views without performing a close-range, high-resolution investigation of every object, a robot will need to operate over a long period of time. Our dataset must therefore have both long total duration and long temporal extent: that is, it must repeatedly view the world,

(a) An observation of the table, with objects.

(b) An observation of the table, without objects.

(c) An observation of the counter, with objects.

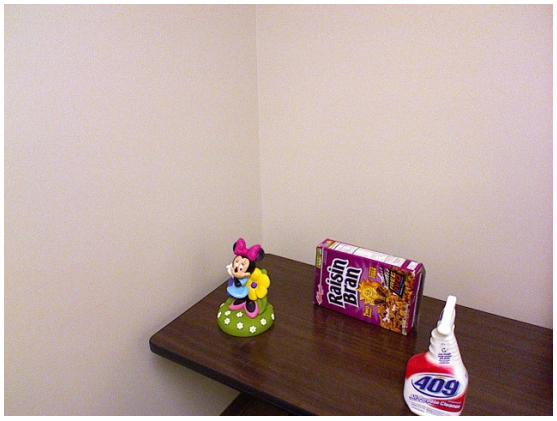
(d) An observation of the counter, without objects. The objects seen here never moved, and (under the mobile objects definition) are therefore not detected.

FIGURE 3.4: The four phases of the medium run. Two scenes (table and counter) are observed, both with and without objects. All four of the objects seen in (a) are seen in both locations.

and these repeated observations must be spread out over a long period of time.

Other obligations

If the *only* goal of a robot is to discover objects, it will spend its time locating objects, and then examining them at close range and high resolution (e.g. the Mobile Objects dataset, where "locating" was done manually, or Krainin et al. (2011), where the robot picked up the objects). A general-purpose robot can expect to have many responsibilities; some amount of time may be available



(a) The first of the four places investigated.

(b) The second of the four places investigated.

(c) The third of the four places investigated.

(d) The fourth of the four places investigated.

FIGURE 3.5: Examples from the large run (Section 3.2). All seven unique objects can be seen. Some objects appear in several locations; others, only one. All four places were also seen without objects (not shown).

for object discovery, but not all. Therefore, our dataset should include both "accidental" object observations and more focused intentional object observations.

Our Willow Garage dataset was taken with these desiderata in mind. Like the Mobile Objects dataset, it was collected on a PR2. Unlike the Mobile Objects dataset, no attempt was made to enforce a particular model of object behavior, and the robot navigated autonomously, rather than being teleoperated.

Willow Garage (a two-dimensional map of which can be seen in Figure 2.1) is a fairly generic office environment. It is roughly 48 m by 61 m, and has a total free area of roughly 143 m^2. It includes the generality we hope for: long corridors, large open rooms, various static obstacles, and difficult navigation problems.

The Willow Garage dataset consists of 67 runs through Willow Garage, spaced out over six weeks. Roughly speaking, three runs were taken each weekday: one in the morning, one in the early afternoon, and one in the evening. Robot availability (and the other disturbances inherent to working in a busy indoor environment) kept us from maintaining a flawless schedule. These runs were taken in one of two ways:

Passive runs

In a passive run, the robot is given a set of fixed waypoints (seen in Figure 3.6) which were chosen by hand. The robot then autonomously navigated to each of these waypoints in turn, pausing at each one. The waypoints remained fixed over the course of the experiment, but the dynamic nature of the environment (and the unpredictable nature of navigational planners) led to a variety of robot trajectories. (In extreme cases, the planner even gave up entirely on reaching certain waypoints.) The morning and evening runs each day were passive runs.

It should be noted that these runs did contain a small form of active search. During navigation, the space directly in front of the robot is generally devoid of objects (because the robot is actively searching for empty space through which to navigate). To increase the number of objects in our data, we had the robot actively point its head at tables and other horizontal surfaces as it drove by, up to a maximum yaw angle off-center. As the PR2 does not use the head for navigation, this is a reasonable optimization, but is not a necessity.

Rescan runs

In a rescan run, the robot was given a list of the objects that were automatically

FIGURE 3.6: The waypoints used in the Willow Garage dataset. The waypoints are represented as squares, and the color of each square denotes the order in which the robot was instructed to visit them, with red for the first waypoint, and blue for the last. (After visiting the last waypoint, the robot was instructed to return to the first, at which point the run completed.) The lines also help make clear the order in which the waypoints are visited. Note that the robot navigated in the empty space (white pixels), not along the lines themselves! Note that this Figure matches Figure 2.1.

discovered on the previous passive run. For each object, the robot navigated to the location from which that object was last viewed, and then observed it directly for several seconds.

Each run contains the following data:

Localization

To keep both the robot and discovered objects in a shared coordinate frame, the robot kept itself localized during each run. Specifically, we recorded the moment-by-moment best localization estimate, but not the state of the AMCL particle filter. In addition, we stored the current facing angle of the robot's head; when combined with the robot's calibration parameters, this allowed us to reconstruct the shared-coordinate-frame coordinates of any point returned by the Kinect.

Laser rangefinders

The data from both the base (two-dimensional) and chest (three-dimensional) laser rangefinders were recorded for completeness. These data make it possible to use our dataset for experiments in laser-based SLAM, in both two and three dimensions.

Kinect

Most importantly, we record both RGB and depth data from the PR2's head-mounted Kinect. To keep the data storage requirements from ballooning out of control (and to keep the datarate from outrunning the hard drive), we throttled the RGB-D pairs to 5 Hz. Both the RGB and depth data were recorded at 640×480 pixel resolution.

The 67 runs total 326GB (before lossless compression), 66,185 seconds, and roughly 40 km in distance traveled, for an average of roughly 4.9GB, 16.5 minutes,

and 600 m per run. To our knowledge, this is the largest dataset of repeated robotic observations of an environment that is publicly available. Other Kinect datasets (i.e. Janoch et al., 2011; Lai et al., 2011) focus on objects rather than repeated observations. Like the Mobile Objects dataset, the Willow Garage dataset is available for download. See `http://www.ros.org/wiki/Papers/IROS2012_Mason_Marthi`.

Recall our desiderata: *size, duration,* and *other obligations*. The size of Willow Garage itself (combined with our waypoints covering the area) satisfies the first. The length of each run, combined with the total number of runs (and the six-week duration of the entire experiment) covers the second. Finally, other obligations: we divided our data gathering into "passive" and "rescan" runs to address this issue. A passive run corresponds to the common case, when a robot is engaged in some other activity, and observes objects "by accident." The rescan runs are a simple form of active search, one that was chosen to support change detection (Section 4.3).

4

Software System

In the previous chapters, we described the problem of discovering objects in a robot's environment. At one extreme, the set of objects is small enough (or the accuracy requirements are so exacting) that training a recognizer for every object is worthwhile. At the other extreme of the recognition spectrum the set of objects is so large that supervised training is totally impractical. While our research focus is on the pure-discovery case, we also want to support the in-between case, where some objects are common enough (or important enough, or hard-enough to discover) to deserve supervised training. Training supervised recognizers from unsupervised data is an interesting direction for future work; see Section 7.1.

The research contribution of this document is in object discovery, but we have taken care to situate this work in the larger context of semantic maps, and of deployable robotic systems. As a result, our software layer explicitly supports the mixed discovery-and-recognition scenario described above, and supports operating at scale: not just on a robot, but on a robot in a large, general environment over a long period of time. This chapter describes our system, ambitiously named `megaworldmodel`, which is available at `http://www.ros.org/wiki/megaworldmodel`.

4.1 Design Goals

To support object discovery and semantic mapping (with both discovered and manual labels), our system requires:

Data Persistence

Consider our problem: discovering objects in a large environment over a long period of time. Suppose that an object is seen at some location $\mathbf{p}$, but $\mathbf{p}$ is rarely visited; the robot may only visit $\mathbf{p}$ once a week, or once a month. Because these visits are rare, it is important that we support storing all of them. In addition, the mobile objects assumption (and the related problem of change detection) require that we maintain enough information to know if an object has moved; the interval between first seeing an object and first seeing it move can be extremely long. Therefore, our software must support *persistence*, both of raw perceptual data (RGB-D pairs, robot poses, etc.) and of other information like segmentations, object identities, and semantic labels.

Multiple perceptual pipelines

As noted earlier, object recognition is a vast field of study. In any given scenario, which supervised recognizers are used (if any) will depend on the environment and the objects likely to be encountered. Therefore, we must support multiple perceptual pipelines, each of which starts with a frame and produces some type of "object" output. (Note that object discovery is included in this definition.) We must support running these pipelines in parallel.

Generality of "object"

We may be running many recognition and discovery algorithms in parallel, and each algorithm may define "object" in a different way: as a segment or labeled pose, for example. Our representation of "object" must therefore be

very inclusive, without falling into the trap of being so general that "object" carries with it no shared data whatsoever.

On- or off-line operation

Finally, our software should be able to run online, and on a robot. However, object discovery and recognition are not guaranteed to be online algorithms, so the system must also support performing some (or all) of its operations offline, and cleanly interleaving on- and off-line operations.

The `megaworldmodel` fulfills each of these goals. It is implemented using the Robot Operating System (ROS) framework, described by Quigley et al. (2009). ROS itself is described below, for context; our system is then laid out in detail in Section 4.2.

4.1.1 ROS

ROS is a general-purpose software framework for writing robotic software. As our system leverages ROS capabilities at almost every level, we provide an introduction here. ROS grew from the software-engineering challenge of the STAIR robot (described by Ng et al. (2007, 2008)), which included a wide variety of sensors, effectors, and algorithms. Today, ROS is the *de facto* standard software layer for academic robotics research, and is gaining ground in industrial robotics and robotics education.

In earlier robotics frameworks every component of the system, including hardware drivers, perception algorithms, and high-level execution planners, were linked into the same executable. This approach makes communication between the various components of the system very tricky: if the laser rangefinder is reporting data at 40 Hz but the localization algorithm runs at only 15 Hz, the programmer must specify the synchronization logic between these two subsystems. When scaled to the dozens or hundreds of things happening on complex robot, this process becomes not only

extremely difficult, but extremely brittle: every pairwise interaction requires extra work. Furthermore, linking everything into one process makes the entire system only as stable as its least-stable component: a segmentation fault in one single component will bring the entire system down.

ROS addresses this problem by dividing a system into *nodes*, each of which is a unique process. To facilitate communication between nodes, ROS defines *ROS messages*, a description language for data structures (in short: booleans, numeric types, strings, other messages, and arrays of any of these). For example, the ROS message type for laser rangefinder data is defined thusly:

```
std_msgs/Header header  # A nested message including a timestamp
  uint32 seq
  time stamp
  string frame_id
float32 angle_min   # A 32-bit floating-point value.
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32[] ranges   # The range data returned by this scan.
float32[] intensities
```

Once a message is defined, code-generation tools in ROS produce code in several languages (C++, Python, Common LISP, and optionally Java) to work with the message. To the programmer, a message looks like the appropriate language-specific data structure (in C++, for example, the message above is a class called `LaserScan`). The primary operations on messages (beyond the construction and mutation expected from any data structure) are serialization and deserialization, to support the network operations described below.

Consider a ROS node that implements a driver for a laser rangefinder. This node serves, in effect, as a bridge: it performs whatever operations are necessary to read data from the hardware, and then packages those results up into the appropriate ROS message. To get this message to any interested nodes, the driver node then *publishes* this message. ROS provides a system for inter-node communication called a *topic* which is in effect a network data bus. When a node publishes a message, ROS serializes that message and transmits it to any nodes that have *subscribed* to that topic. Each topic is named (for example, the PR2 base laser described in Section 3.1 publishes on a topic called `base_scan`), and strongly-typed: `base_scan` transmits only `LaserScan` messages. Finally, topics are many-to-many: as many nodes as want to can publish, or subscribe, to each topic.

The multiprocess communication model solves several problems. First, it eases synchronization: each node need only deal with its own subscriptions, and the queuing policy can be specified per-subscription. Second, process-level isolation makes the entire system robust to problems with a single node; ROS can be told to simply start new copies of any nodes that crash. As topics can be subscribed or published to at any time, these replacement nodes will join in gracefully.

Finally, the many-to-many nature of ROS topics makes possible a wide variety of useful graph topologies. As an example, consider the problem of recording the datasets described in Chapter 3. In a single-process model, each data producer would have to be modified to also write its data to a log file. In the ROS model, the data-recording tool (a node called `rosbag`) simply subscribes to every relevant topic and records the arriving messages to a file, requiring no code modification whatsoever. Playback is equally simple: rather than start up the nodes that encapsulate hardware drivers, `rosbag` open a log files and publishes the stored messages in order; nodes subscribing to those topics have no way of knowing that they are receiving log data, not live robot data.

Our software system is deeply connected to ROS; we implement our algorithms in ROS nodes, transmit our data using ROS messages, and store our logs in the `rosbag` format. Furthermore, the PR2 is a native ROS platform: the localization, navigation, and hardware we use are all used through ROS.

4.2 Architecture

Our system is laid out as a pipeline (which can be seen in Figure 4.1). We begin by using ROS to gather localized RGB-D data (that is, frames, as defined in Section 1.2). We define a ROS message called `ImagePair` to carry frames. In principle, object discovery tools might want access to two- or three-dimensional laser data or other sensor information. In practice, the localized RGB image, depth image, and point cloud[1], in an `ImagePair` cover the input requirements of every object-level algorithm that interests us.

These `ImagePair` messages are then published to one or more *recognizers*, each running as a ROS node. We use the word "recognition" extremely loosely here: "recognitions" include the output of both supervised recognition algorithms (with class labels) and the segmentation algorithms described in Chapter 5, which return unlabeled sets of pixels. To support both models of "recognition", we define a `Recognition` message:

```
# Defines the ImagePair this recognition came from.
string imagepair_id

# Which recognition algorithm generated this message? Code that
# processes Recognition messages needs to know this to know how to
# interpret the data below.
string recognizer
```

[1] The point cloud is stored implicitly: an `ImagePair` message contains the intrinsic parameters of both the RGB and depth cameras, so point clouds can be generated by projecting the depth image into three dimensions.

```
# Which object class does this belong to? The empty string means
# "unknown".
string class_id

# Which object instance is this? Tracking instances allows us to
# support recognizers that can tell instances apart (say, tools that
# have both a concept of object identity and a concept of object
# location).
int32 instance_id

# If our recognizer returns a subset of the image (that is, is a
# segmenter), it fills in the pixels field.
# Pixel-based segmentation. This is a list [row, col, row, col, ...].
uint32[] pixels

# RGB and D images are not necessarily the same size; this flag tracks
# whether the pixels above are RGB pixels or depth pixels. (The
# ImagePair can be queried directly to determine the sizes of the images.)
bool pixels_in_depth_image

# Here's the other possibility; an object posed in 3D space, rather than
# a list of pixels. The exact details of what this means (where the
# object's coordinate frame is anchored, for example) depend on which
# recognizer you use; the recognizer field will tell you which one you
# used. PoseStamped is a built-in ROS message type; a
# six-degree-of-freedom pose with a timestamp.
geometry_msgs/PoseStamped pose

# A numerical score representing the quality of a recognition. What
# this value means is specific to the algorithm that generated
# this message.
float32 score
```

A recognizer takes in an `ImagePair` and publishes one or more `Recognition`

messages, which are received by *associators*. An associator (itself a ROS node) clusters `Recognition` messages into object instances and object classes. Should the recognizer return a class identity (as a supervised recognizer might), this clustering would be a trivial operation. In the object discovery case, association is one of the fundamental problems we must solve. Associators publish a `Clustering` message, encapsulating their results.

As described above, our system satisfies only two of our desiderata: supporting many recognizers and having a general concept of "object." To gain persistence and on-and-offline operation, we introduce an intermediate layer: a database. A ROS library (called `warehousewg`) provides the ability to store ROS messages in a MongoDB "NoSQL" database. This database not only makes our ROS messages persistent on disk, but allows us to to associate with them arbitrary key-value metadata, and then to query over these metadata. As an example, consider the `recognizer` field in our `Recognition` message. We associate the key-value pair {`recognizer : X`} (where `X` is the name of the recognition algorithm) with each `Recognition` message. This allows us to trivially perform queries like "Give me every `Recognition` found by the `supporting_planes` recognizer" or "Give me every `Clustering` produced by the `stationary-objects` associator."

We insert the database between every stage of our pipeline, as can be seen in Figure 4.1. Note that nodes do not pass ROS messages from one layer to the next; instead, each layer pushes messages to the database, and each following layer pulls relevant messages from the database. In the worst case, this forced persistence adds only one network operation: copying the message to the database.

A simple optimization eases even that small overhead in practice. When a recognizer adds a `Recognition` message to the database, the database transmits a *notification message* on a fixed topic. This message contains only metadata, and is therefore only a few bytes. Associators listen on the notification topic to learn when

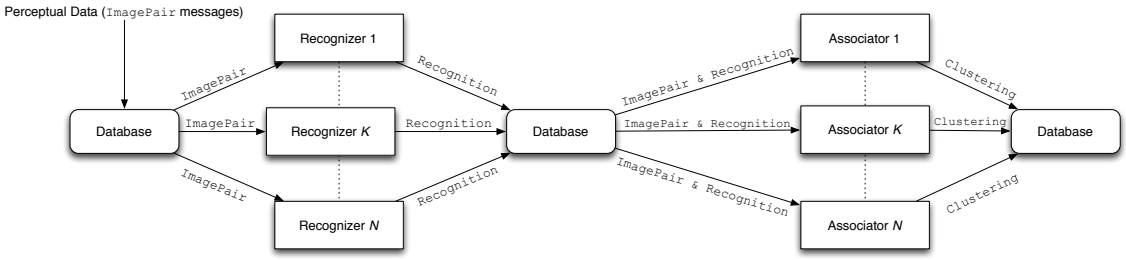


FIGURE 4.1: System architecture. Arrows denote interprocess communication, labeled with the appropriate ROS message type. This figure elides the database notification topics for clarity. Note that there is only one database process; each database box represents a different type of stored message (a "collection" in MongoDB nomenclature).

to pull a new `Recognition` message from the database. Note that not every type of associator can process the output of every type of recognizer; associator nodes can use the contents of the notification message to decide not to pull the recognition message at all, saving a transmit-deserialize step.

This pull-when-needed behavior is used at each step of the pipeline, and also permits chaining together multiple recognizers or associators; each only processes results generated by its immediate parent.

Finally, the database-backed pipeline model also makes it easy to interleave online and offline operations. For example, the frame registration step described in Section 6.2.1 is computationally expensive. It can be run online, but in our experiments it was most convenient to run it in batch mode, processing every frame in a single pass, rather than as they arrive. In the database model, this is easy. With the database running, we start only the nodes we need (rather than the whole pipeline) and then generate and transmit the notification messages that "drive" the registration step. The registration algorithm then processes these in turn, and stores the result in the database.

4.3 Applications

The database-backed model we describe has another advantage: it provides an easy backend for writing applications that query the state of the semantic map. In our first paper (which used an earlier version of the system, but was still database-backed), we implemented two applications: *change detection* and *semantic querying*. Change detection is the classic security guard problem: "When changed since the last time I observed this scene?" Semantic querying is more general: it includes queries over any label in the database. To avoid the need for recognition, we limited ourselves to labels that could be directly computed from the data. consensus color (computed by binning HSV pixels into named bins and picking the largest one), approximate diameter (by projecting a segment's point cloud onto the x, y and z axes and choosing the largest one), approximate shape (by taking a histogram of local curvatures over the cloud, and declaring the object "planar" if the curvature is close to flat at most points, and "curved" otherwise) and finally a place label such as "cafeteria" (computed from the segment's pose by comparing against a set of predetermined locations).

The results of one such semantic query can be seen in Figure 4.2. Both applications were implemented using a web front end, which (as of this writing) is available at `http://worldmodel.willowgarage.com`. *It bears repeating that the perceptual pipeline used in these examples is an early version, circa our work in Mason and Marthi (2012), and underperforms the final version described in this document.*

The applications in the first paper were instance-only: there is no model of object class. This approach limited them to simple applications, but its simplicity allowed them to scale to large and general environments. Our work in object discovery was partly motivated by a desire to add more-powerful capabilities without scaling down.

(a) (b) (c) (d) (e) (f)

FIGURE 4.2: Every result for the semantic query "Medium-sized things in the cafeteria" in a particular run. See Section 4.3 for details. *Please note that the perceptual pipeline used to generate these results predates, and underperforms, the final version described in this document.*

5

Segmentation

We factor object discovery into two subproblems: segmentation and association. This chapter discusses segmentation, and our approach to it. Association is the subject of Chapter 6.

Our contribution to the segmentation problem is two algorithms: one focused on segmenting objects that change location ("mobile objects", detailed in Section 5.2)) and one focused on objects that don't ("stationary objects", detailed in Section 5.3). When applied to a dataset where objects move (Section 3.2), the mobile objects algorithm demonstrates 84.4% precision and 87.5% recall. When the stationary objects algorithm is applied to a dataset without guaranteed object motion (Section 3.3), it too demonstrates high precision: 85.6%. (Due to the scale of the dataset used, calculating recall in this case is not feasible; see Section 5.3.1). This performance demonstrates that the first half of object discovery is possible.

In Section 1.2, we defined a *segment* as a connected subset of the pixels in a frame. Traditional image segmentation seeks to partition an image into segments that correspond to "similar" areas. However, there is no agreed-upon criterion for evaluating the quality of a segmentation. In a classic figure from Felzenszwalb and Huttenlocher

FIGURE 5.1: An example of a scene to segment. In this case, there are four objects: a detergent bottle, an egg carton, a ziploc bag box, and a Minnie Mouse coin bank. A perfect segmentation would include four segments (one per object), and each segment would contain exactly those pixels that belong to the object. Note that a single object can have a variety of textures and colors: for example, the detergent bottle has substantial areas of blue, yellow, and white.

(2004), an image of a baseball player is segmented into several components, including his hand, glove, arm, head, and body. As each segment has self-consistent appearance, this segmentation is in some sense "correct"; however, if the goal is to segment out baseball players (which are self-similar at a semantic level), the segmentation is a failure.

As a result, classical segmentation fails to capture our interest in objects. Rather than partition the image into "similar areas", we want to return segments that correspond to objects. Objects do not necessarily have self-similar appearance (see Figure 5.1, among other figures in this document), nor are similar areas necessarily

objects (again Figure 5.1, noting the walls and tabletop). For us, a segmentation algorithm takes in a frame, and returns zero or more segments, each of which should correspond to an object. This allows us to define a "good" segmentation as one with the following properties:

High Precision

The *precision* of a segmentation is the percentage of the returned segments that are actually objects. In our case, perfect precision means that every returned segment is an object (or part of one), and zero precision means that every returned segment is something other than an object.

High Recall

The *recall* of a segmentation is the percentage of the objects in the scene that are returned. In our case, perfect recall means returning a segment for every object that appears in a frame, while zero recall means returning none of the objects that appear.

Single-segment objects

Finally, the segments returned should correspond as closely as possible to *complete* objects. That is, each object should have only one segment, and that segment should include the entire object.

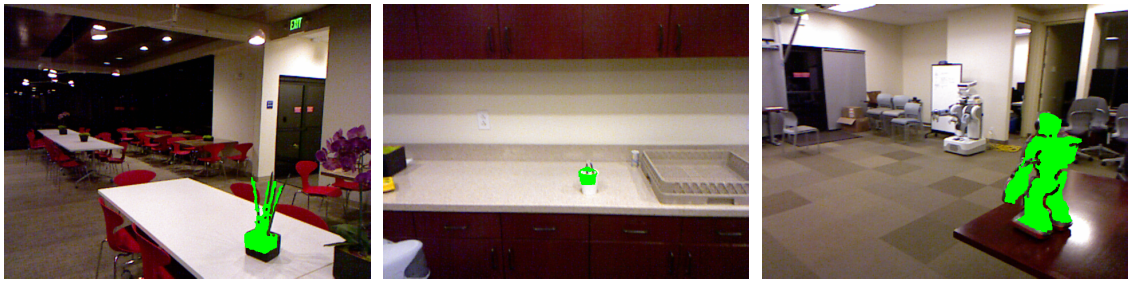
With these goals in mind, consider a single segment (example segments can be seen in Figure 5.2). That segment will fall into one of five categories:

Perfect segment

A single segment corresponding to an entire object, as in Figure 5.2a and Figure 5.2c. An omniscient algorithm would return nothing but perfect segments.

Partial segment

A segment may correspond to less than an entire object, as in Figure 5.2b and



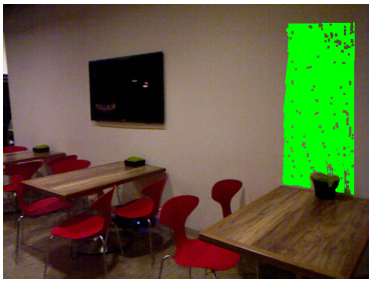
(a) A houseplant.

(b) A coffee cup.

(c) A bipedal robot. (A different instance of this object can be seen in Figure 1.2.)

(d) An undersegmentation.

(e) An RGB-D synchronization error.



(f) A non-object.

FIGURE 5.2: Examples of segmentation results, including correct and incorrect segments. *This figure is best viewed in color.*

Figure 1.2a. Such a segment contributes to high precision and high recall, but not single-segment objects.

Oversegmentation

Oversegmentation is when an object (in a single frame) is broken into several segments. This is distinct from a partial segmentation, in which an object has only one segment in a given frame, but that segment does not cover the entire object. Like partial segments, this contributes to high precision and high recall, but not single-segment objects.

Undersegmentation

In an undersegmentation, one segment contains pixels from more than one object, as in Figure 5.2d. Undersegmentations can also occur when a segment is

primarily object pixels, but includes some non-object pixels. Undersegmentations contribute to none of our desiderata.

Non-object

A non-object is a reported segment that does not correspond to any object at all, as in Figure 5.2f. Non-objects are unmitigated errors, and contribute to none of our desiderata.

Finally, it is possible for a segmentation to miss an object entirely, not reporting a segment where one should occur. This is a *false negative*.

Our segmentation algorithms divide segmentation into two steps, named:

Partitioning

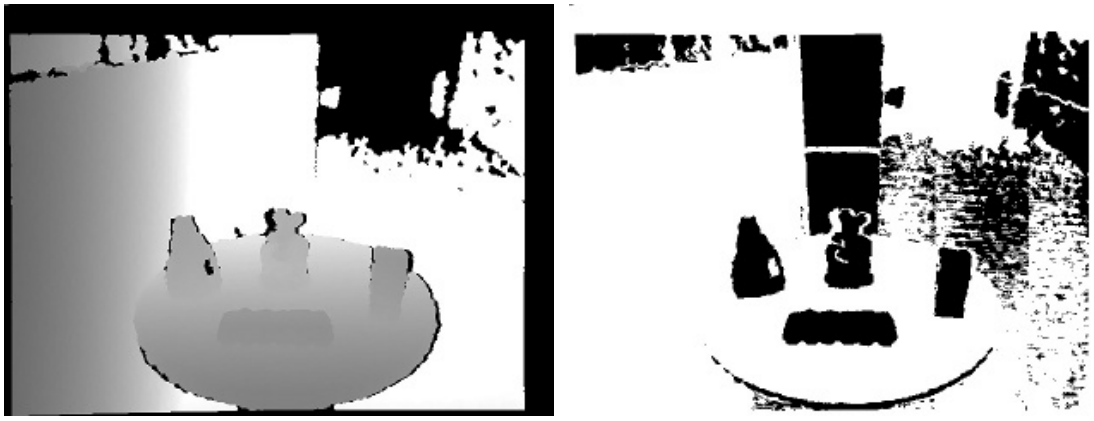
Partitioning determines which parts of the image *might* be objects, and which are certainly not. This partitions the image into zero or more might-be-object *regions*, and marks the rest of the image as "non-object". Non-object pixels are ignored in the next step. Partitioning is the subject of Section 5.1.

Selection

Given a partitioning, *selection* determines which regions, if any, actually correspond to objects. Here, we lean on our heuristic definitions of "mobile" and "supported" objects (from Section 1.1). Selection selects a subset of the might-be-object regions returned by the partitioning step and returns them as our segments. Our selection algorithms are discussed in Section 5.2 and Section 5.3.

5.1 Partitioning

As we use the term, *partitioning* takes in a depth image and partitions the pixels in that image into image *regions*. We expect a true partitioning of the image: every pixel is either in a region, or is marked as a non-object.



(a) The depth image. (b) The computed mask M.

FIGURE 5.3: The partitioning pipeline. (a) is the depth image; the corresponding RGB image can be seen in Figure 5.1. The computed mask image M is (b). 0-valued (black) connected components of (b) are our regions. Several things are noteworthy here. First, the wall behind Minnie (parallel to the image plane) is not masked off. Both the failure to mask and the horizontal masked stripe are due to problems with plane fitting, and are discussed in Section 5.1.1. The speckling on the floor is caused by the same issue. The next step (selection) correctly chooses not to return any of these errors as regions. See Figure 5.6 for more about this scene.

We present two selection algorithms in later sections. Both use the same partitioning algorithm, which leverages the "supported object" heuristic in a slightly unexpected way. Under the heuristic, objects must rest on some large, horizontal surface (ignoring the floor, which trivially supports everything). Therefore, the pixels on the surface itself are not object pixels, nor are pixels on flat, vertical surfaces (because the object would fall off). To this, we add the assumption that sharp depth discontinuities signal the edges of objects; while not true in the case of cluttered scenes, experience shows these to be rare in practice (see Section 5.3.1). Planes are detected and masked out (see Section 5.1.1), as are depth edges. The partitioning algorithm is detailed in Algorithm 1, and an example input-output pair can be seen in Figure 5.3.

Note that the set of 0-valued connected components ("regions"), when combined with the set of 1-valued pixels ("non-object points"), form a partitioning of the

```
Input  : A n × m depth image D.
Output: A n × m binary mask image I.
I ← new n × m image of zeros
// Masking off planar surfaces
S ← the set of planar surfaces in D
foreach s ∈ S do
    foreach pixel p in s do
        I(p) ← 1
    end
end
// Masking off depth edges
for r ← 1 to n − 1 do
    for c ← 1 to m − 1 do
        for i ← −1 to 1 do
            for j ← −1 to 1 do
                if |D(r, c) − D(r + i, c + j)| > ε then
                    I(r, c) ← 1
                end
            end
        end
    end
end
```

ALGORITHM 1: Computation of the mask image for partitioning (Section 5.1).

image.

5.1.1 Computation of Planar Surfaces

Our algorithm relies heavily on the ability to detect planar surfaces. Our early work, as well as the surface detection for the mobile objects experiments, found planes using the Random Sample Consensus (RANSAC) algorithm of Fischler and Bolles (1980). Briefly, RANSAC fits a model to the data by constructing random models, and then checking their fit to the data. In our case, that means projecting the depth image into three dimensions (that is, a point cloud), randomly selecting three points, solving for the plane that they define, and then checking every point in the cloud to see if it fits the model. Models that fit large numbers of points are kept, and those points are marked as being a member of a planar surface.

While accurate, RANSAC has two important limitations. The first is speed: each random model must be checked against every (remaining) point in the cloud, leading to $O(n)$ work (where n is the number of points; 640×480 in our case) per model. In practice, this meant that plane extraction took two to three seconds per frame. The second issue is connectivity: the three-point model of a plane does not enforce the requirement that the points on the plane be connected. As a result, points which fit the model, but are not part of the supporting surface, will be marked as part of the plane. This can be seen in Figure 5.3b: a horizontal stripe of disallowed (white) points can be seen on the wall behind Minnie, and on the right of the image in the distance. These points fit the model of the table in the foreground, but are not part of it. (A connectivity criterion could be applied in a post-processing step.)

Because of these issues, the stationary objects results use the plane-finding algorithm described by Trevor (2012). This technique finds planes by computing connected components in the depth image. Two pixels are deemed to be connected ("part of the same plane") if the orientation of their surface normals differs by less than a threshold, and their distance from the camera differs by less than a threshold. In our experience, this technique gives qualitatively similar results while running at roughly 10 Hz.

5.2 Selection of Mobile Objects

Gibson (1986) defined a "detached" object as a something that "can be moved without breaking our rupturing". As we have noted before, detachability is a clean definition of "object", but is hard to measure passively.[1] Therefore, in Section 1.2, we extended detachability slightly, to what we call a *mobile object*: an object that is not only detached, but actually observed to move. We limit ourselves to semi-static

[1] In the active setting, a robot could apply force to parts of the world, for example with a manipulator, and see if it can be moved. In principle, this could require an extremely strong robot. This has been investigated by Bersch et al. (2012), for example.

worlds, in which the object does not move on video: instead, the object is observed in a static scene, and then later *not observed* in another view of the same scene. We leverage this "now you see it, now you don't" property to select those regions that demonstrate mobility.

Herbst et al. (2011a,b) also use mobility as a cue for object discovery. Their technique is dense: they must store three-dimensional metric information about every surface in case it becomes relevant. Octree-based mapping, (e.g. Mason, 2010; Fairfield, 2009; Hornung et al., 2013), could also be used, but any dense technique is necessarily sensitive to small localization errors, particularly at the resolutions necessary to detect objects.

Rather than store a dense representation, we propose a map of *visual features*: RGB image points with a highly recognizable appearance. As a sparse subset of the pixels are selected by our visual feature detector, our map is automatically sparse. To support the temporal analysis necessary to detect mobility, our map also stores a timestamp for each feature.

5.2.1 Visual Features

A *visual feature* is an image patch (in our case, in an RGB image) that is easily "recognizable" in some sense. For a feature f, this takes the form of $\mathbf{x}_f$, the image coordinate of the feature, and $\mathbf{d}_f$, a *descriptor*. The descriptor is a vector that summarizes the appearance of the image patch. A feature is "re-recognized" when a new feature f' is found such that the distance between $\mathbf{d}_f$ and $\mathbf{d}_{f'}$ is below some threshold. As defined, descriptors can be computed for any image patch, not only "interesting" patches. To limit meaningless results (for example, all patches that are entirely white will have the same descriptor, but this is rarely a useful fact), algorithms for computing visual features first detect *interest points*, which are patches likely to generate discriminative descriptors. Descriptors are then calculated for

these points, and the algorithm returns a list of $(\mathbf{x}, \mathbf{d})$ pairs. A wide variety of visual feature algorithms exist: the seminal approach (for recognition) is the SIFT feature of Lowe (1999). Riding a recent surge of interest in visual feature design, our algorithm uses the ORB feature of Rublee et al. (2011), which was primarily chosen because it can be computed very quickly without resorting to GPU implementations.[2] Although we have not investigated other features in detail, nothing in our technique is fundamentally incompatible with SIFT or other features. The ORB descriptor is based on a set of binary comparisons, meaning that the descriptor is a bit vector (of 256 dimensions), and the correct distance measure between descriptors is the Hamming distance.

5.2.2 Sparse Feature Map

Consider a single frame, and a single ORB feature f computed from the RGB component of that frame. Using the value of pixel $\mathbf{x}_f$ in the depth image, we can project f into three dimensions (in the camera's coordinate frame), and then use the robot's localization estimate to pose f in the global coordinate frame. Our sparse feature map is composed of a set of such posed features, each with the timestamp of the frame that contains it. Let M_1 denote such a map, just before the arrival of a new frame.

Given M_1 and the new frame, the question we wish to answer is "Which map points *should have* been observed, but were not?" Such points are then treated as candidate on-object points. We filter the points in M_1 according to a series of criteria to determine which points these are; that process can be seen in Figure 5.4, and is detailed below.

The use of an RGB-D camera makes the geometric part of this question straight-

[2] Due to their large power and cooling requirements, powerful GPUs are not easily deployed on mobile robots.

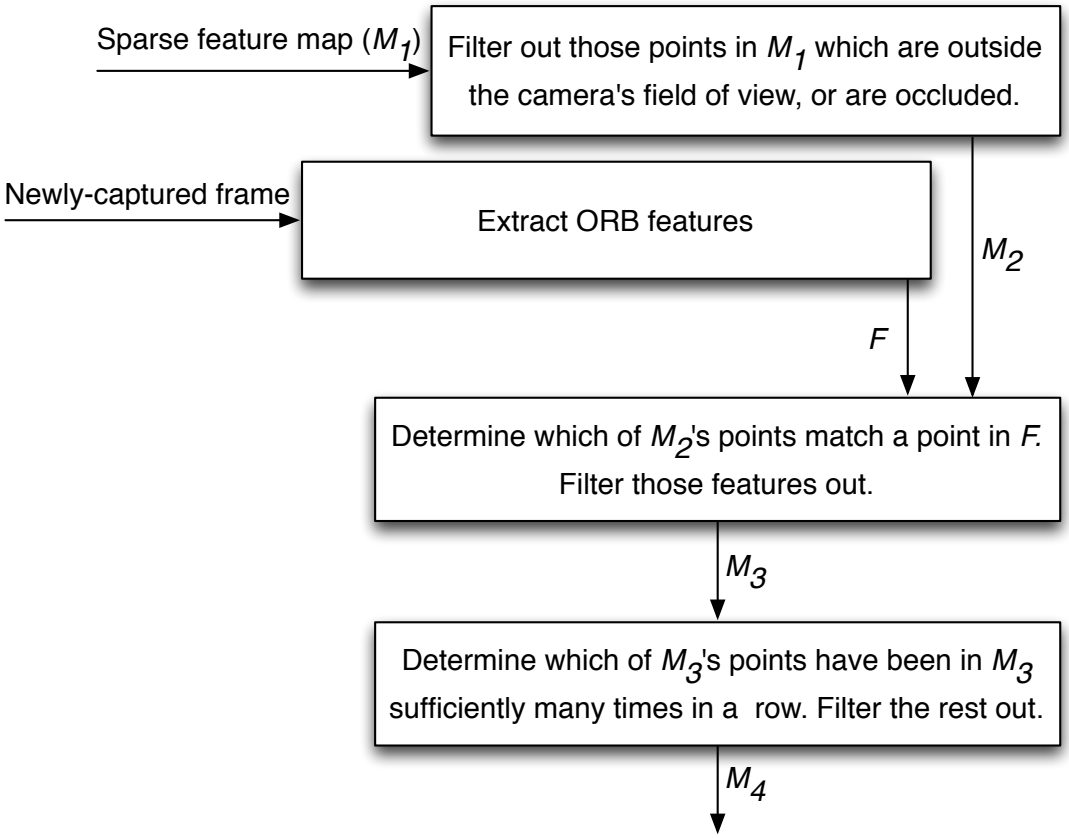
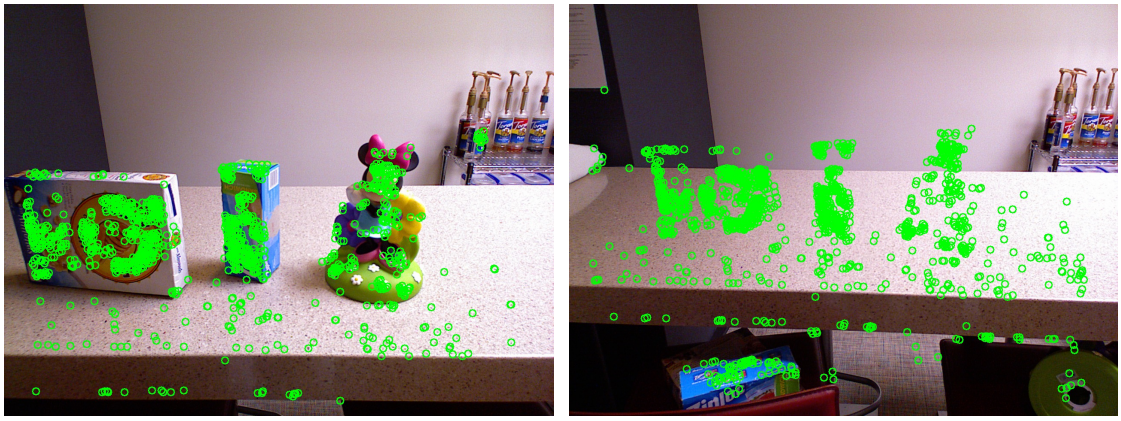


FIGURE 5.4: The Mobile Objects feature-processing pipeline (Section 5.2.2). Features in the map are filtered to determine which should have been observed in the current frame, but were not.

forward. Given M_1 and the robot's current localization estimate, we can project every point p in M_1 into depth-image coordinates. The camera z-coordinate (forward out of the image plane) can then be checked against the depth image: if the z-coordinate in the depth image is smaller (that is, closer to the camera) than the z-coordinate of p, then p is occluded in this frame. We cannot reason about the disappearance of occluded points, so they should be skipped. This filtering leaves us with a set of points $M_2 \subseteq M_1$ that should have been observed in this frame.

Next, we must determine which points in M_2 were actually observed. We compute visual features in the current RGB frame, and project them into 3D. We define a spatial threshold s and a descriptor-distance threshold d. A new feature f is deemed



(a) An image, with the features in M_1 drawn on.

(b) A later observation of the same place, with M_4 drawn on.

FIGURE 5.5: An example of detecting visual feature disappearance (Section 5.2.2). Note that some feature clusters are not on objects; however, they are on planes (or on regions with too few points), and therefore do not generate a segment.

to match a map feature m if f is within both the s and d thresholds. Applying these thresholds to each element of M_2 gives us a set $M_3 \subseteq M_2$ of features that should have been observed, but were not. See Figure 5.5 for an example. (At this stage, the newly-computed features are also added to the map.)

A feature map is an efficient representation only if it is not too heavily populated with useless features. Like any product of an image, visual features are subject to noise, motion blur, and (in our case) accidental occlusion due to localization error. These issues can lead to missed matches: features that should have been observed, but erroneously were not. Because we use negative feature detections as our cue for segmentation, we wish to avoid *false negatives*, even at the risk of false positives. We introduce two techniques to retain only stable features. First, we enforce *temporal stability*. We require that a feature be observed (seen for the first time, or matched) for k frames in a row before it is added to the map. This helps to filter those features that are highly sensitive to image noise. We also enforce temporal stability on the matching side: to count as a true negative, a feature must fail to match (that is,

be in M_3) k times in a row; this helps to account for transient misses due to image noise.

Second, we introduce the concept of a *feature cluster*. Because objects will be seen from several different viewpoints, the same point in 3D space may have many different appearances, and may generate a several different feature descriptors. To account for this, our feature clusters store several descriptors. The cost for a new feature f to match against a cluster c is then the minimum of the descriptor distance between f and any of the descriptors in c. To update feature clusters, we introduce a second spatial threshold, the *integration threshold* i. If f is within i of c, we add f's descriptor to c's set. Importantly, we do not perform a descriptor-distance check first: the goal of a feature cluster is to capture the variability of descriptor values due to viewpoint, and requiring a close descriptor match would defeat this. Applying the temporal stability criteria to the features in M_3 gives us $M_4 \subseteq M_3$, which we pass as input to the next stage of the segmentation algorithm.

Our feature clusters draw some inspiration from the HOC descriptor of Pirker et al. (2010), which also seeks to handle viewpoint effects in visual features.

In our experiments, we set the spatial threshold s to 5 cm, the descriptor distance threshold d to 150, the temporal stability threshold k to 5, and the integration threshold i to 2 cm. These parameters were determined by hand, and the same values were used for every experiment.

5.2.3 Segmentation of Mobile Objects

When a set of candidate object points is identified (M_4, above), these points are passed to the mobile objects selection algorithm. The goal is to label the RGB-D data in an *earlier* frame that correspond to the missing object in the *current* frame. Consider every frame (starting with the very first frame in the run) in which *any* feature in M_4 was observed. Let $f_1 \ldots f_n$ denote the partitionings of these frames,

sorted by timestamp. Given these inputs, Algorithm 2 (also discussed below) returns the segments that caused the features that have now failed to appear.

```
Input  : M_4 and f_1 ... f_n.
Output: A set of segments S.
S ← {}
foreach f ∈ f_1 ... f_n do
    r_1 ... r_k ← the regions in f
    C ← a new array of k integers, all 0
    foreach m ∈ M_4 such that m was observed in f do
        p ← the region in f in which m falls
        if p = no region then
            continue
        end
        C[p] ← C[p] + 1
    end
    for i ← 0 to k − 1 do
        if C[i] > ε then
            S ← S ∪ {r_i}
        end
    end
end
```

ALGORITHM 2: The Mobile Objects selection algorithm. In short, each "disappeared" feature in M_4 votes for the regions in which it was observed. Those regions with enough votes are returned.

Consider a single partitioning $f \in f_1 \dots f_n$; it contains a set of regions, $r_1 \dots r_k$ (see Figure 5.3b for an example of a partitioned depth image). Recall that every feature in M_4 *should have* been observed in the current frame, but was not. Consider a single feature $m \in M_4$ that was not observed in the current frame, but *was observed* in f's frame. That is, m was caused by some object that was present when f's frame was captured, and has since disappeared. Using the three-dimensional coordinates of m, we can project it into the image plane of f, thereby determining its pixel coordinates in f. This tells us which (if any) of the $r_1 \dots r_k$ (that is, which potential object) generated it.

Given the M_4 generated by a just-arrived frame, we perform this analysis for each

(a) (b)

FIGURE 5.6: Features selecting regions in mobile objects segmentation. The depth image and partitioning corresponding to these RGB images can be seen in Figure 5.3. Figure (a) shows a frame with the features that have been "walked back": those features seen *in this frame* that are seen to disappear in a later frame. These features are assigned to their corresponding regions, and those regions with sufficiently many are returned. In this case, those regions can be seen in false color in (b). *This figure is best viewed in color.*

$m \in M_4$, and keep track of how many features have landed in each region. Those regions with enough features are declared to be objects, and returned.[3] As example of what this looks like can be seen in Figure 5.6.

5.2.4 Performance

We validate the mobile objects algorithm on the (helpfully-named) Mobile Objects dataset, described in Section 3.2.

To provide a baseline against which to compare our segmenter, we hand-segmented each run in the Mobile Objects dataset. For every occurrence of an object in our data, we manually find the bounding rectangle (in image space) and assign a label according to the object name. Our automatic segmentations are not, in general, rectangular (or even convex). To compute the overlap between a hand segmentation h and an au-

[3] In principle, this threshold should adapt to the size of the region in question, as small regions (including far-away objects) are likely to produce fewer features than large regions. We performed no such normalization for our experiments.

TABLE 5.1: Mobile Objects segmentation performance (Section 5.2.4). See Table 5.3 for another algorithm applied to this dataset.

Run	Small	Medium	Large
Hand segments	270	394	419
Auto segments	270	396	423
True positives	270	392	357
False positives	0	4	66
False negatives	0	2	51
Precision	100%	98.9%	84.4%
Recall	100%	99.4%	87.5%

tomatic segmentation a, we first find the bounding rectangle r of a. We then declare h and a to match if $\text{Area}(h \cap r) \geqslant 0.5 \cdot \text{Area}(r)$ and $\text{Area}(h \cap r) \geqslant 0.5 \cdot \text{Area}(h)$. This 50–50 overlap criterion is common; Kang et al. (2011) use it (for non-rectangular segments), for example. We can then compute precision and recall scores for each dataset. See Table 5.1 for the results.

5.2.5 Object Appearances

As described and tested above, our algorithm detects object disappearances, but disappearances are not the only kind of semi-static object motion: objects can also *appear*. Our algorithm can be applied to appearances as well. Consider the sequence of events of a disappearing object: it is seen at some time t, and then not-seen at some later time t'. An *appearing* object has the opposite behavior: it is not-seen at t, and then seen at t'. However, our feature clusters carry timestamps; to detect appearing objects, all we need to do is *run time backwards.* When this in done, an object seen at time t' and not-seen at time t "disappears", so our algorithm can detect it.

The Mobile Objects dataset intentionally includes only object disappearances, not object appearances; to test on appearances, we would have to reverse the dataset, and then run the algorithm backward on the reversed data, which is identical to running the disappearance algorithm on the original data.

5.3 Selection of Stationary Objects

For all of its benefits, object motion is an onerous requirement. While objects that are moved are likely to be of note, many useful or interesting objects stay put. As an example, consider a computer monitor: monitors are moved only very rarely, but are a common item in office settings, and well-worth recognizing.

To handle stationary objects, we press forward with the idea of a "supported object". In partitioning, the heuristic was used to mark non-object points. Here, we make the natural observation that points *supported by* supporting surfaces are likely objects. We approximate "supported by" to mean "points which are above, but not too far above, a supporting surface."

In any particular frame, the surface that supports an object may not be entirely visible. To solve this problem, the stationary objects algorithm integrates supporting surfaces over time. We store a global *plane state*, representing the set of known supporting surfaces. Each surface is represented by the convex hull of its three-dimensional points; these points are forced to be coplanar (and horizontal), but include an altitude. After each partitioning step, any horizontal planes found are added to the plane state, and any overlapping planes are merged. One can imagine non-convex supporting surfaces; representations such as the α-shapes described by Edelsbrunner and Mücke (1994) could be used instead. In practice, our supporting surfaces are tables, counters, and shelves, nearly all of which are convex (in fact, most are simply rectangular). Stationary objects selection is detailed in Algorithm 3.

Because partitioning operates solely in the depth image, both selection algorithms encounter difficulties in cluttered environments like that seen in Figure 5.2d. Undersegmentation errors in these cases were surprisingly rare in the Willow Garage dataset; see Section 5.3.1. (The Mobile Objects dataset, which was built by hand, contains no cluttered scenes.) This indicates that (at least in our environment), such

```
Input  : A depth image D, and the global plane state G.
Output: A set of segments S, and an updated plane state G'.
S ← {}
// R is the set of regions; P is the set of horizontal planes.
(R, P) ← partition(D)
// Update global plane state.
G' ← G
foreach p ∈ P do
  | Add p to G', (recursively) merging planes as needed.
end
// Find supported points.
S ← {}
C ← project-to-point-cloud(D)
foreach point c ∈ C do
  | foreach plane g ∈ G' do
  |   | // z_a is the vertical coordinate of some point a.
  |   | if c is within g and z_g ⩽ z_c ⩽ z_g + ε then
  |   |   | r ← element of R where c occurs
  |   |   | S ← S ∪ {r}
  |   |   | continue
  |   | end
  | end
end
```

ALGORITHM 3: Stationary objects selection. In our experiments, ϵ was 6 cm.

object configurations are rare. Note that given sufficiently dense data (like that used by Karpathy et al. (2013)) objects can be extracted from clutter directly. However, our data-collection regime (detailed in Chapter 3) gives us data that are too sparse to permit such an approach. Like cluttered scenes, stacked objects would also pose a problem for stationary object selection, as we make no allowance for objects that support other objects. Such objects would appear as undersegmentations.

5.3.1 Performance

We apply the supporting planes segmentation algorithm to both the Willow Garage dataset (described in Section 3.3) and the large run from the Mobile Objects dataset (Section 3.2). Because small localization errors can cause the supporting planes to

grow too large, parts of the walls can end up being marked as object (as they are "supported" by an over-large surface). Our approach already masks out large planes in any orientation, so this "wall object" error (an example of which can be seen in Figure 5.2f) is fairly rare. To further minimize this problem, we reset the global plane state after each run. While this was done as an optimization, it also introduces robustness against supporting planes that change location, or disappear entirely: for example, a table that is removed should not be allowed to support objects after it disappears!

Hand-labeling every object in every frame of the Willow Garage dataset is infeasible; the dataset contains 331,034 images, of which 14,815 are kept by the velocity filter. As a result, we instead consider those segments we do find, and report on their quality in Table 5.2. To our surprise, undersegmentations were very rare: only 36 segments, or 2.4% of the total. More information about these segments can be found in Section 6.2.3, where they are associated.

We also apply this algorithm to the large run from the Mobile Objects dataset, and report the results in Table 5.3. Note that these numbers differ from those reported for mobile objects segmentation. In particular, the total number of segments is much lower (103, rather than 423). This is due to a change in the velocity filter. In the mobile objects case, we kept every frame in which the robot's velocity was low enough. To save on processing, the stationary objects algorithm kept only the first frame of each period of low-velocity travel. As a result, the precision scores are not directly comparable; nevertheless, the stationary objects algorithm performs well.

TABLE 5.2: The types of segments discovered by the stationary objects selection algorithm (Section 5.3.1) on the Willow Garage dataset. "Good" segments correspond to all or part of an object, and include partial segments, oversegments, and perfect segments. "Bad" segments are everything else. Undersegmentations, which we would expect from cluttered scenes, are surprisingly rare. See Table 5.3 for the results of this algorithm on the Mobile Objects dataset.

	Count	**Percentage of Total**
Undersegmentations	36	2.4%
Non-objects	183	12%
Total "Bad" segments	219	14.4%
Total "Good" Segments	1300	85.6%
Total Segments	1519	100%

TABLE 5.3: The results of running stationary objects selection on the large run from the Mobile Objects dataset. See Table 5.1 for the results of mobile objects selection. See Table 5.2 for the results of this algorithm on the Willow Garage dataset.

	Count	**Percentage of Total**
Undersegmentations	2	1.9%
Non-objects	8	7.7%
Total "Bad" segments	10	9.7%
Total "Good" Segments	93	90.3%
Total Segments	103	100%

6

Association

Our second object discovery subproblem is *association*: grouping segments into meaningful clusters. In Section 1.2, we defined two types of clusters: *instances* and *classes*. Instances correspond to objects *in locations*, while classes correspond to types of objects, independent of location.

Our work in semantic mapping demonstrated that instance-only association could be put to useful, but limited, ends. To move forward, we need to perform association at the level of classes. Classwise association is a clustering problem: we seek to cluster segments into classes. A difficulty is that k, the true number of object classes (and therefore clusters), is not known.

Buoyed by our success in using visual features for segmentation (Section 5.2), we investigated modeling the appearance of each class with visual features. This led us to a probabilistic model (based on a Dirichlet process) that performs clustering without the need for k. This algorithm is discussed in Section 6.1. When run on the segments extracted from the Mobile Objects dataset, this approach performs well: 86.2% precision and 72.2% recall (compared to 68.9% and 59.9% for k-means clustering).

However, clustering using appearance alone presents a problem: not every segment has useful visual structure. In fact, many of the objects in the Willow Garage dataset are small and textureless (or nearly so). Figure 6.1 contains two good examples of this problem: as these objects are uniformly white, little to no appearance information is available. With this in mind, we developed a second algorithm (detailed in Section 6.2) that combines appearance, shape, and pose to perform clustering. Despite using a simpler, non-probabilistic formulation of the clustering problem, this technique demonstrates remarkably good results on the Willow Garage dataset: 98.7% precision and 71.8% recall. (Note that the algorithm was tuned to maintain very high recall; this is discussed in detail below.)

The goal of this document is to prove that object discovery is possible in general environments, and that robots help. These results, in combination with our segmentation results, prove exactly that.

6.1 Probabilistic Association

Our probabilistic association algorithm relies on a classic observation: segments that belong to the same class should have *similar appearance*. Defining "similar appearance" is a difficult problem for several reasons:

Lighting

The raw pixel values in an image (and therefore on a segment) are sensitive to changes in lighting, even for a fixed camera observing a fixed object. The specularities in Figure 3.5c (as compared to the other images in Figure 3.5) provide an example, and more subtle variations occur throughout our data.

Occlusions

If an object is partly occluded, some of its appearance is simply missing. If the missing pixels can be picked out, matching could proceed on the visible pixels.

However, knowing (in general) which parts of an object are visible requires having already recognized the object!

Viewpoint changes

Finally, many objects appear different from the front and the back (or the side). As a result, accurately matching a view from the front of an object to a view from the back requires training data that include both views.

We describe the appearance of each segment using a bag of visual words (detailed below), and then cluster the segments using a Dirichlet process mixture. As this technique uses only object appearance (not object location), it discovers only classes, not instances. As the Mobile Objects dataset contains well-separated objects seen from relatively few viewpoints, it poses too easy an instance-level problem to be a useful test case. Nevertheless, applying the pose-based reasoning described in Section 6.2.2 to perform instance-level association would be straightforward.

6.1.1 Bag of Visual Words

See Section 5.2.1 for a review of visual features and notation.

For the reasons detailed above, we do not describe our segments using raw pixel values or raw feature descriptors. Instead, we use a *bag of visual words* to describe each segment. A bag of visual words (BOW) model (introduced by Sivic et al. (2004) and elaborated on by Sivic and Zisserman (2006, 2008)) begins by computing visual features. It then seeks to limit the effect of small image changes on the descriptor values by defining a set of exemplar descriptors ("visual words"). Each computed feature is then replaced with its nearest neighbor from the set of exemplars. Because of this replacement, only "large" changes in descriptor values (a single feature's descriptor must change enough to move it to a different nearest exemplar) are reported. This limits the effects of image noise or viewpoint changes.

The performance of BOW models is sensitive to the choice of visual words, so we learn ours from the data. Given a robot run (or multiple runs) we perform segmentation (Chapter 5) on every frame, which returns a list of segments S. For each $s \in S$, we compute new ORB features for that segment alone. We concatenate the results for all segments into a single list of descriptors that covers the entire run.[1] Importantly, we do not reuse the ORB features computed for selection (Section 5.2). The selection features were computed over an entire frame, and therefore include both on- and off-object points; the goal here is to represent the variety of appearance in objects, not the entire environment. Our segmentation-association factorization helps us here, as it gives us a concrete definition of "object" (that is, a segment) to work with.

Given the list of descriptors from the entire run, we perform K-means clustering to learn our visual words. In our experiments, we learned a set W of 250 visual words.

Next, we consider each segment in turn. For each segment, we quantize its visual features into their visual words, and return a list $\mathbf{w}$ of those words. As the visual word descriptors are known in advance, we perform an optimization: $\mathbf{w}$ stores the indices of the visual words, not their descriptors. As a result, $\mathbf{w}$ can be thought of as a histogram, with one bucket per word. This histogram is our representation of the appearance of a single segment.

Given these histograms, we must now cluster them into classes. Clustering and association are well-studied problems, but our lack of k (the number of desired clusters) limits our options. Rather than apply cross-validation or other techniques to find k, we have chosen a probabilistic model based on Dirichlet processes which does not take k as a parameter.

[1] The selection of visual words presents an opportunity for (semi-)supervised training. If certain objects can be exhaustively scanned ahead of time, their visual words could be added to the training set before the data-collection run begins.

6.1.2 Dirichlet Processes

A Dirichlet process (DP), as introduced by Ferguson (1973), is a probability distribution that is particularly well-suited to clustering problems in which the number of clusters is not known in advance. A DP has two parameters: a *base measure* G_0 and a *concentration parameter* (sometimes called a *new cluster rate*) $\alpha > 0$. A sample from the DP is itself a probability distribution over the same space as the base measure.

The probabilistic model used in clustering problems is an extension known as a Dirichlet process mixture. In a DP mixture, each cluster has an associated probability distribution from which its elements are drawn. As an example, consider clustering the elements of some finite set E.[2] In this case, each cluster could be a multinomial distribution over the elements of E. (In this case, index i of distribution j is the probability that cluster j generates element i.) The base measure G_0 is a prior distribution over these cluster distributions; in our example, the base measure would be a Dirichlet prior on multinomial distributions over E. A draw from the DP then yields a specific countably infinite mixture G of multinomial distributions. Each observed element is then generated by sampling a mixture component from G, and then sampling from that component's multinomial distribution. Two elements are in the same cluster if they were generated by the same mixture component. Given the elements to be clustered, the unobserved quantities are the DP sample G and the identities of the clusters from which each element was drawn. In practice, all we want is the partitioning of elements into clusters. Neal (2000) presents several Markov-chain Monte Carlo (MCMC) algorithms that will sample approximately i.i.d. clusterings from the posterior conditioned on the observations. We can then use these samples to answer our inferential questions.

[2] Note that this an example, not the clustering problem we are trying to solve.

6.1.3 Generative Model

Intuitively, our generative model is this: the "world" (modeled as a DP) generates object classes (modeled as multinomial distributions) which generate segments (which we can measure). More specifically, our base measure G_0 is a Dirichlet distribution over our set W of visual words. Therefore, each mixture component ("class") is a multinomial distribution, also over visual words. An observed segment is generated by choosing a component from the DP, and then sampling independent visual words from the component's multinomial distribution. Given a set of observed segments, our goal is to assign each one to the cluster that generated it.

6.1.4 Inference

We use the collapsed sampler described by Neal (2000). The algorithm maintains a set of samples, each of the form $(x_1, \ldots x_M)$, where M is the number of segments. Each x_m is a class identifier; in our case, a positive integer. Initial samples may be generated in any way; we use $(1, 2, \ldots, M)$. At each iteration, the algorithm flips the m^{th} component, where m repeatedly sweeps over $(1, \ldots, M)$. As each component is a class ID, flipping the m^{th} component assigns segment m to some existing class, or to a new class. The probability of assigning segment m to class c is proportional to

$$W_{-m,c} \int F(s_m, \phi) dH_{-m,c}(\phi). \tag{6.1}$$

Here, $W_{-m,c}$ is the number of visual words on segments other than m currently assigned to class c. $H_{-m,c}(\phi)$ is the posterior distribution over multinomial distribution ϕ based on the prior Dirichlet distribution G_0 and observations of these visual words, and F is the likelihood of the words in segment s_m given ϕ.

The probability of assigning m to a new class (one not previously assigned to any segments at all) is proportional to

$$\alpha \int F(s_m, \phi) dG_0(\phi) \tag{6.2}$$

where α is the concentration parameter of the DP. At each iteration, the quantities above are computed for all existing object IDs and the new object ID, then normalized. The resulting discrete distribution is sampled from to find the new assignment of m.

6.1.5 Performance

We evaluated our DP model on the Mobile Objects dataset (Section 3.2). For each of the three runs, we ran the sampler for 5000 scans, where one scan performs one flip for each segment. To determine the best clustering given the samples, we could simply take the most likely clustering. However, our samples are high-dimensional, and any given sample is therefore unlikely to appear very many times. Instead, we take the per-segment mode: each segment is assigned the object ID that it was most-commonly assigned in the sample set. This technique assumes that object IDs are stable across samples, which means it will not work for arbitrary sampling schemes. However, in our Gibbs sampling scheme, it is extremely unlikely that every segment belonging to a single object will flip to a new object ID all at once.

We wish to evaluate the end-to-end performance of our system, but segments that do not correspond to objects do not have a correct cluster available for assignment. (Nor do they necessarily have consistent appearance, so a single "non-object" cluster is not feasible either.) We therefore run our system end-to-end (meaning that non-object segments are clustered), and then mark each non-object segment as "invalid". Invalid segments are ignored for the purposes of analyzing association performance; see Section 5.2.4 for segmentation performance.

Next, we define:

True Positive

A pair of segments (i, j) is a *true positive* (TP) if i and j are from the same class in reality, and are in the same cluster.

False Positive

A pair of segments (i, j) is a *false positive* (FP) if i and j are not from the same class, but are in the same cluster.

False Negative

A pair of segments (i, j) is a *false negative* (FN) if i and j are from the same class, but are not in the same cluster.

We compute TP, FP, and FN for all pairs of segments (i, j) (for $i \neq j$), and report two values:

Precision The *precision* of a clustering is the fraction of returned positives that are true positives: $TP/(TP + FP)$.

Recall The *recall* of a clustering is the fraction of the *total positives* that are correctly returned: $TP/(TP + FN)$.

For each run, we report the precision and recall. For a baseline, we compared our results on the large run against K-means clustering. For K-means, we treated each segment's visual word counts as a (normalized) multinomial distribution, and used the total-variation distance. Unlike our algorithms, K-means was provided with the true cluster count (seven). Results can be seen in Table 6.1.

6.2 Deterministic Association

Our DP model, while highly general, is limited by its focus on object appearance. Many of the segments discovered in the Willow Garage dataset are too small to

TABLE 6.1: DP clustering performance compared to a K-means baseline on the large run.

Run	S	M	L	K-means on L
Unique objects	2	4	7	...
Auto segments	270	396	423	...
Invalid segments	0	4	36	...
Precision	100%	100%	86.2%	68.9%
Recall	100%	100%	72.2%	59.9%

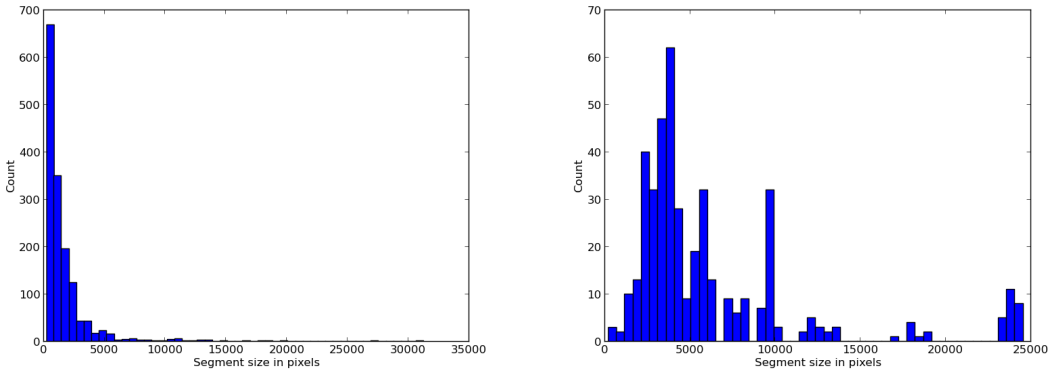
contain sufficient visual structure for feature matching (or are large enough, but contain no visual structure at all; see Figure 6.1 for an example of both problems.) As can be seen in Figure 6.2, the median size of a segment is 1018 pixels, roughly the area used to compute one ORB descriptor.[3] Finally, the DP model ignores both object location (meaning that it cannot cluster instances) and three-dimensional object shape, which is a potentially powerful cue. To better leverage these data, we present a second unsupervised clustering algorithm, which is better suited to the Willow Garage dataset.

Our approach is based on graph connectivity. Each segment s is a node in a graph G, and two segments are connected by an undirected edge if they satisfy a set of similarity criteria described below. The connected components of the resulting graph are our clusters. Like the DP model, this approach does not require specifying k; unlike the DP model, it is not probabilistic, and does not inherit the robustness that probabilistic models can bring. In particular, our reliance on "hard" assignments makes our approach brittle to false-positive edges: in the worst case, a single false-positive edge could lead to the misassociation of very many segments. As our experiments demonstrate, we successfully avoid this problem in practice.

[3] Or four SIFT descriptors. Note that both ORB and SIFT use square patches, and our segments are not necessarily square.

(a) (b)

FIGURE 6.1: An example of the difficulty posed by using only appearance in matching segments. Both segments seen here are flat white, but should not be matched. *This figure is best viewed in color.*



(a) Histogram for the Willow Garage dataset. The smallest, median, and largest segments have 250, 1,018, and 31,287 pixels, respectively.

(b) Histogram for the large run from the Mobile Objects dataset. The smallest, median, and largest segments have 208, 4,130, and 24,541 pixels, respectively.

FIGURE 6.2: A histogram of the sizes of the segments in the Willow Garage dataset, and the large run from the Mobile Objects dataset. Both histograms have 50 bins.

6.2.1 Instance Association

The need for instance-level association can arise in two ways: viewpoint changes and partial segments. Because we extract segments on a per-frame basis, two different frames that observe the same object necessarily create two segments. In addition, segmentation can oversegment or generate partial segments. Instance association helps us recover from these segmentation results, which are "correct" in that they contain objects, but do not fulfill the single-segment objects criterion described in Chapter 5.

As instances are defined by both an object class and a location, we can use our localization to determine if two segments overlap, and therefore whether they are part of the same instance. However, localization is only accurate to within a few centimeters (which is often a substantial fraction of the size of an object), so pure-localization techniques suffer from both false-positive associations (when two different segments are "smudged together" by localization mistakes) and false-negative associations (when two segments are pulled apart).

A more subtle problem is replacement: consider a segment s, created from an object at some position $\mathbf{p}$. Next, consider what happens if that object is removed, and a different object is placed at $\mathbf{p}$, and the robot observes the new object, generating a segment t. The segments s and t are not from the same class, and should not be associated. However, location-only techniques will fail, as s and t overlap. (Interestingly, the DP model would handle this case correctly, although the Mobile Objects *segmentation* would not.)

Overlap also introduces a subtle, and extremely difficult, problem: what if s and t are in the same class? In one sense, s and t should be treated as different instances: after all, they are physically different objects. In another sense, they should be the same instance: as the robot did not observe the replacement, it has no way

of knowing that s and t are different. In this work, we take the second view, and consider associating s and t to be correct. Solving the more-complex form of the problem would require a notion of a "unique instance", where different elements of the same class can be distinguished.

Our first paper (Mason and Marthi, 2012) does instance-level association, but using an extremely simple algorithm. Given two segments s and t, their point clouds (in the global coordinate frame) are projected flat (that is, the vertical axis is ignored). The segments are deemed to be in the same instance if the convex hulls of these two clouds overlap. This approach is sensitive to localization error, and makes no effort to avoid the replacement problem. The performance of this algorithm is included in Figure 6.6.

Our complete algorithm improves upon the convex hull technique (for instance association) in two ways: by correcting for localization error, and by using three-dimensional, not two-dimensional, spatial overlap. While these cannot entirely solve the problem, they do limit its scope to extreme localization errors (which are rare) and extremely exacting replacements (also rare). This is presented in Algorithm 4, and the details are discussed below.

Consider two (different) segments s and t, and the RGB-D frames that generated them, f_s and f_t. To determine if s and t are part of the same instance, we need to check them for spatial overlap. However, doing so accurately requires correcting for localization error. In full generality, correction would be done using a SLAM algorithm, but SLAM adds considerable implementation and computational complexity.[4] We do something simpler: pairwise alignment.

We begin with the convex-hull-overlap analysis described above. If the hulls overlap at all, we proceed to the next step: otherwise, we do not add a graph edge

[4] The potential SLAM problem is particularly severe, as it would require performing SLAM not just within a run, but between runs, thereby registering the *entire dataset.* The difficulty of registering this much data is exactly what limits Herbst et al. (2011a,b) to small environments.

Input : Two segments s and t, and their frames f_s and f_t.
Output: A boolean: do s and t pass the spatial-overlap check?

```
// An optimization:  start with a two-dimensional overlap check.
h_s ←two-dimensional-convex-hull(s)
h_t ←two-dimensional-convex-hull(t)
if !intersects(h_s, h_t) then
    return false
end
// Proceed with full overlap check.
trans ←icp(f_s, f_t)
// Align t with s according to ICP.
t' ←apply(trans, t)
// Do volumetric overlap.
v_s ←voxelize(s)
v_t ←voxelize(t')
overlap ← |v_s ∩ v_t|
return v_s ⊆ v_t or v_t ⊆ v_s or (overlap ⩾ ϵ|v_t| and overlap ⩾ ϵ|v_s|)
```

ALGORITHM 4: Our improved three-dimensional overlap check. In our experiments, $\epsilon = 0.1$. As more than one segment can occur in each frame, and ICP is run between frames, the ICP results are cached.

between s and t. This step is a filter, and is strictly an optimization, as the full alignment step is expensive.

Given non-zero convex-hull overlap, we proceed to align the full point clouds of f_s and f_t using the Iterated Closest Point (ICP) algorithm introduced by Besl and McKay (1992), specifically the implementation in PCL (Rusu and Cousins, 2011). We initialize ICP using the transformation estimate provided by localization. As ICP is run between full frames (each of which may contain several segments) the results are cached in the database. (This demonstrates another advantage of our database-backed model: our ICP implementation uses no ROS IPC infrastructure, but can rely on the database for persistence anyway.)

Next, we compute the three-dimensional overlap between s and t. Computing the overlap between the three-dimensional convex hulls is sensitive to noise: consider the case of minor undersegmentation where a single point on the background has

been included. The resulting hull would be forced to include the volume between the object and background, which could be large relative to the size of the object. We compute an approximation to volumetric overlap: voxel grid overlap.

Consider a dense grid of voxels, each 1 cm on a side, covering the entire environment. For each three-dimensional point p in s, we compute which voxel contains p, and add that voxel to a set. We do the same for t, and our three-dimensional overlap is the number of voxels in the intersection of these two sets. We do not require that the set of voxels be connected: in our point-on-the-background example, we would introduce only one extraneous voxel. The voxelization operation is extremely fast: to convert a point (whose coordinates are in meters) into voxel coordinates, all that is needed is to multiply by a constant (in this case, 100: converting from meters to centimeters) and convert from a floating-point representation to an integer. Because the segments have small spatial dimensions, we can store just those voxels that occur, rather than allocating the dense grid described above, making the entire operation fast and memory-efficient. Let S and T denote the voxel sets for segments s and t. We declare that s and t overlap if $|S \cap T| \geqslant 0.1|S|$ and $|S \cap T| \geqslant 0.1|T|$ (the 0.1 was chosen by hand). To handle the case of one object being entirely inside another (which can happen when a comparing a partial segment to a complete segment, for example) we also declare an overlap if $S \subseteq T$ or $T \subseteq S$. If s and t overlap, the edge (s, t) is added to our graph.

6.2.2 Class Association

Associating segments into classes ("class discovery") is the harder, but more fundamental problem. Unlike instance association, class discovery cannot rely on location information: segments belonging to the same class can occur in any location. For example, every segment in Figure 1.3 is a member of the class "houseplant," despite appearing in a variety of locations. To discover object classes, we must therefore

consider other information. Our algorithm relies on our two basic assumptions: segments that belong to the same class should have *similar appearance* and *similar shape*.

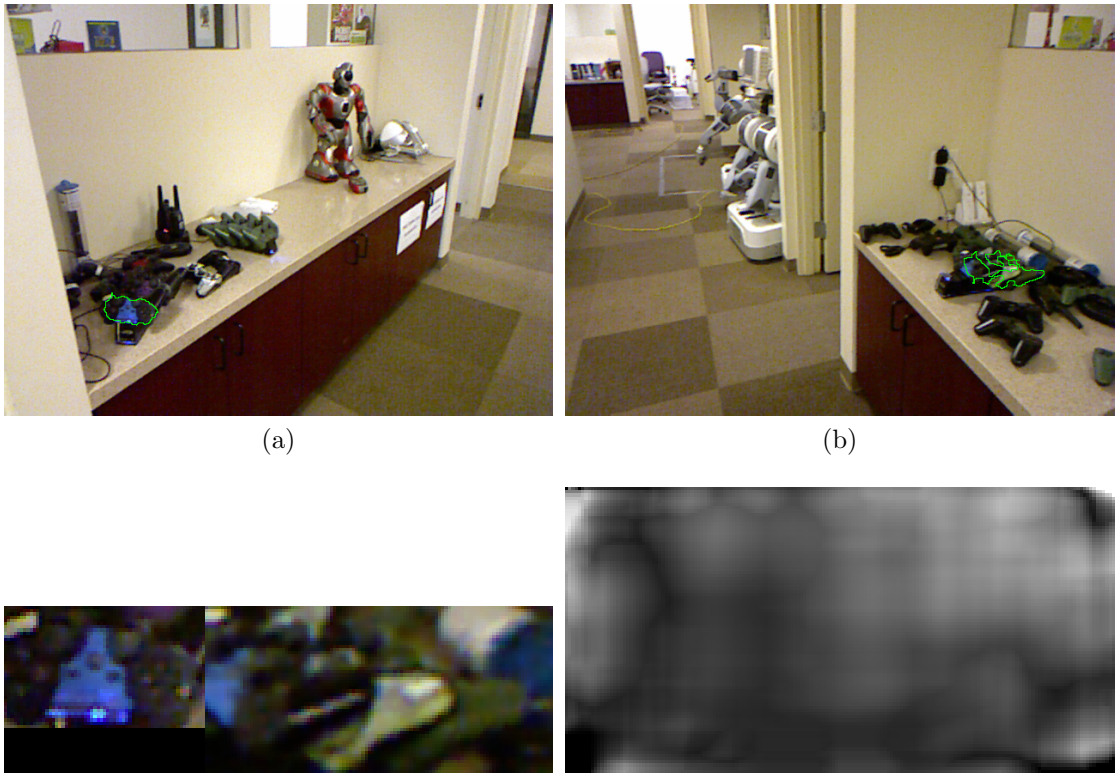
As any two segments in the same instance are necessarily in the same class, we begin by running instance-level association. Class-level association then (potentially) adds more edges to the graph, and our classes are the connected components of the result. Our approach is not hierarchical: we use the same graph for both problems. This lets us avoid the difficult problem of defining the "consensus appearance" or "consensus shape" of an instance.

A common approach for measuring appearance (e.g. Kang et al., 2011) is to measure the distance between color histograms. Because the histogram discards the geometry of the segment, histogram distance has the advantage of being robust to alignment errors. However, it is not robust to certain kinds of partial segments. Consider the two segments shown in Figure 6.3. These are two instances of the same class (a rack of video-game controllers). Because Figure 6.3a is a partial segment, its color histogram is primarily blacks and blues, while Figure 6.3b has a more uniform histogram, including greens and whites. To permit matches in such cases, we perform a search over possible alignments of the two segments, computing a histogram distance at each alignment.

We do this by taking the rectangular bounding box of both segments. The smaller rectangle is then swept across the larger rectangle, and the histogram distance between the overlaps is computed. This process creates a heatmap, as seen in Figure 6.3d. The final appearance distance between two segments is defined as the smallest value in the heat map.[5]

The discussion above omits an important fact: by working directly in pixel space,

[5] Although our implementation computed this value by brute force, our sliding window search is an excellent candidate for the Efficient Subwindow Search algorithm of Lampert et al. (2009), which would greatly accelerate the computation.



(a) (b)

(c) (d)

FIGURE 6.3: Color histograms for appearance-based association. In these figures, we color the outline of the segment (rather than every pixel) to leave the colors visible. Figure 6.3c shows the the "zoomed" versions of the regions, and the resulting heatmap is shown in Figure 6.3d. The appearance cost assigned to this pair is equal to the minimum value in the map. The images shown in (c) are *after* the depth-aware scaling is applied. *This figure is best viewed in color.*

the histogram analysis is sensitive to scale. Consider the bipedal robot seen in Figure 1.2. These are two instances (Figure 1.2a and Figure 1.2b are one instance; Figure 1.2c is another) of the same object, but taken from different distances. As a result, a pixel in one segment corresponds to a different amount of physical space than a pixel in the other segment. To correct for this, we would like to "zoom in" the more-distant segment until we are observing it from the same distance as we are observing the closer segment. Because we have depth information, the distance to each segment is known. Let z_s denote the average distance to the points in segment s, z_t denote the same for segment t, and let $z_s < z_t$. Under weak perspective

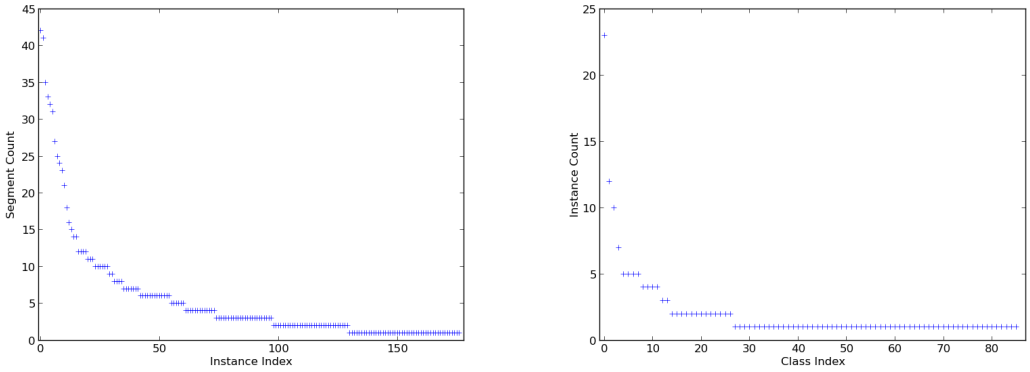
projection, "zooming in" s is equivalent to simply enlarging s by the factor z_s/z_t. We perform this correction before performing the overlap analysis described above. As our camera has finite resolution, we are necessarily interpolating between pixels in the more-distance segment. To avoid matching image-scaling artifacts, we skip entirely those segment pairs where $z_s - z_t$ is greater than one meter.

Our implementation uses RGB histograms with four buckets per channel, compared using the total variation distance.

While effective at generating matches, the histogram technique can fail when a large object with a region of uniform color is compared against a small object of the same color. The small object can "fit into" the larger region, thereby achieving a very low histogram cost. To correct for this, we compute the height, width, depth, and total (voxelized) volume of each segment, and require that they each differ by no more than a fixed ratio. While this is a very simple definition of "shape", it rules out objects of substantially different size.

Finally, we note that even the combination of these techniques can fail. Consider the two segments in Figure 6.1. Both are basically uniform white, but one is a mug, while the other is a bowl. Furthermore, their approximate dimensions are the same. Therefore, we introduce one more cue: a general shape descriptor. We use the Viewpoint Feature Histogram of Rusu et al. (2010), which operates solely in the depth image, and was designed to tease apart similarly-shaped objects. A VFH descriptor operates on the relative orientations of the surface normals of a segment, summarizing them into a 308-element histogram. We compute a distance between segments using the recommended χ^2 distance.

We introduce edges to our graph for those pairs of segments whose histogram cost is below a threshold H, and whose ratio of height, width, and depth are each above a threshold V, and whose shape-cost difference is below a threshold F. We then perform a connected-component analysis on the resulting graph, and deem each



(a) (b)

FIGURE 6.4: The distribution of segments-per-instance and instances-per-class in the segments computed from the Willow Garage dataset. In both plots, the horizontal axis specifies an instance or class, and the vertical axis provides the count. Left off of (a) (for reasons of scale) are one instance with 213 segments and one instance with 94 segments.

connected component to be a class.

See Section 6.2.3 for the results, and Section 6.2.4 for a discussion of the settings for these thresholds and how they were chosen.

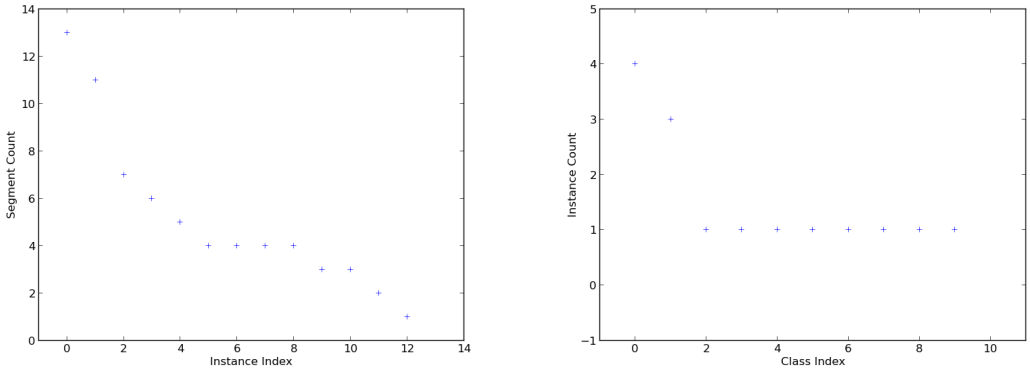
6.2.3 Performance

We evaluate the nonprobabilistic association algorithm on both the Willow Garage dataset and the large run from the Mobile Objects dataset. We begin by manually labeling the instance and class of each segment in both datasets.

In the ground-truth labeling of the segments from the Willow Garage dataset, there are 179 instances drawn from 86 classes. The distribution of segments per instance and of instances per class can be seen in Figure 6.4.

In the ground-truth labeling of the segments from the large run from the Mobile Objects dataset, there are 15 instances and 10 classes.[6] The distribution of segments per instance and instances per class can be seen in figure Figure 6.5.

[6] Note that these ten classes are more than the seven found by Mobile Objects segmentation. As objects are not required to move to be segmented, this is expected.



(a) (b)

FIGURE 6.5: The distribution of segments-per-instance and instances-per-class in the segments computed from the large run in the Mobile Objects dataset. This Figure follows the format of Figure 6.4. The reader will note that most of our classes have only one instance (that is, appear in only one location). Recall that the segments being labeled here are *those found by immobile objects*, not a true ground-truth labeling of the objects in the environment. In this case, two mobile objects were missed entirely.

In the ground-truth labeling our segments s and t can be related in one of three ways:

Disconnected segments s and t are not in the same class, and therefore not in the same instance.

Intra-instance connectivity s and t are in the same instance (and therefore also in the same class).

Inter-instance connectivity s and t are in the same class, but not the same instance.

However, our algorithm can make only two choices: either s and t are in the same connected component, or they are not.

We compute three values: classwise precision, intra-instance recall, and inter-instance recall. Classwise precision is exactly the precision calculated for the DP algorithm.

To compute intra-instance recall, consider every pair (s, t) of segments in the same ground-truth instance. Intra-instance recall is the fraction of these pairs such that s and t are associated by our algorithm. Intra-instance recall tells us how completely our algorithm recovers instances from segments. Inter-instance recall is similar to intra-instance recall, but considers the (s, t) pairs such that s and t are in the same ground-truth class, but different ground-truth instances. Note that all s and t in the same ground-truth class must have either intra- or inter-instance connections (but not both). Inter-instance recall tells us how completely we discover the class structure of the data.

We compute these values over all (s, t) pairs (such that $s \neq t$). We describe our choice of parameters below, and present the results in Figure 6.6.

6.2.4 Parameter Selection

Our association algorithm has three parameters: the appearance threshold H, the spatial threshold V, and the shape-cost threshold F. We tuned our parameters by performing a parameter sweep: for each training set (detailed below), we varied H from 0.01 to 0.55 by steps of 0.01, varied V from 0.5 to 1.0 by steps of 0.01, and varied F from 1 to 300 by steps of 10. The ranges and step sizes were determined by hand. At each step, we recorded the parameter setting that generated the highest inter-instance recall while maintaining a precision (on the training set) of 0.98 or greater. This high precision threshold is required by our use of hard association decisions: a single false-positive association between two segments can lead to two large clusters being incorrectly associated.

Traditional cross-validation is a poor fit for our problem, as a held-out set that is a small fraction of the total data will likely contain very few (s, t) pairs that should be connected. As a result, few potential connections will even be considered, let alone found.

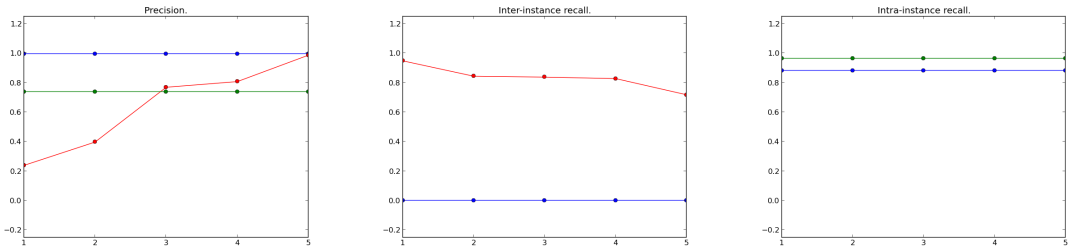
Instead, consider what might happen should we deploy our system in a novel environment. Should performance prove poor, a set of segments would be hand-labeled and used to train new parameters. Should these new parameters not prove good enough, further data would be labeled. (Note that our emphasis on high precision means that the second hand-labeling process could use the automatic labeling as a starting point.)

We simulate this process by sorting our segments by the time of their capture, and then partitioning them into five groups. We perform our parameter sweep training only on group one, then on groups one and two, then groups one, two, and three, and so on. Rather than evaluate on a held-out set, we evaluate on the entire dataset after each training round and report that value. The results of this process on both the Willow Garage and Mobile Objects datasets are detailed below and in Figures 6.6 and 6.8.

Willow Garage

The parameters found by training on the entire dataset (the rightmost data point in Figure 6.6) were $H = 0.21$, $V = 0.84$ and $F = 121$. Over the five examples shown, the difference between the smallest and largest H, V, and F found were 0.04, 0.29, and 30, respectively.

The results shown in Figure 6.6 merit some discussion. First consider the green lines, which correspond to using the two-dimensional convex-hull overlap criterion described in Section 6.2.1. (Because neither the convex-hull criterion nor our improved overlap criterion are subject to a parameter search, both the blue and green lines are constant.) Because of localization error, using only convex hulls should be expected to over-associate, and this is what the low precision score (Figure 6.6a) demonstrates. Similarly, over-association would be expected to lead to high recall, as can be seen in Figure 6.6c. Finally, because the convex hull technique cannot as-



(a) (b) (c)

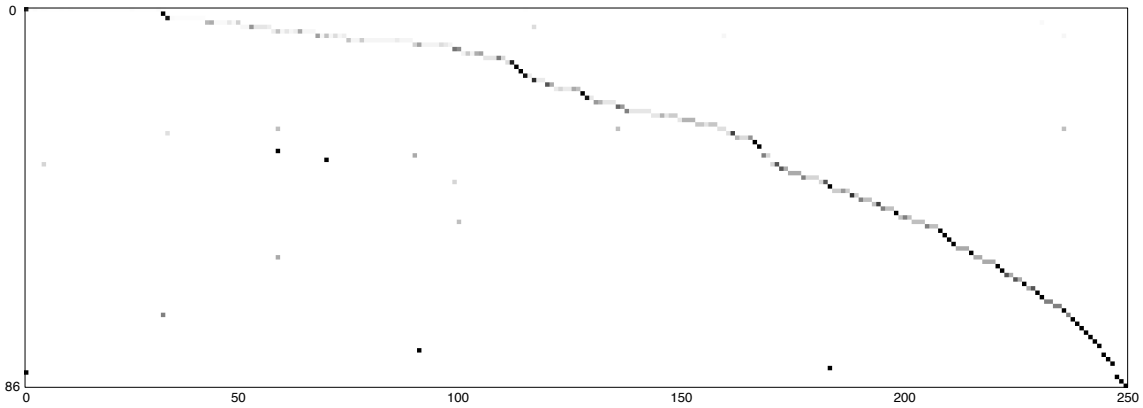
FIGURE 6.6: Results of our deterministic clustering algorithm. (Section 6.2). In each figure, the the blue points correspond to running only instance-level association (Section 6.2.1), while the red points include class-level association. The green points (which are behind the blue points in (b)) represent using only convex-hull instance-level association. The horizontal axis specifies the training set: 1 corresponds to training on the first fifth of the data, 2 to training on the first two-fifths of the data, and so on. Each point is computed by evaluating over the entire dataset. Because we use a single graph for both instance- and class-level association, adding edges generated by class-level association removes the distinction between an instance and a class. Therefore, no red line appears in (c). *This figure is best viewed in color.*

sociate instances in different locations, it has zero inter-instance recall (Figure 6.6b; note that the green line is hidden behind the blue line).

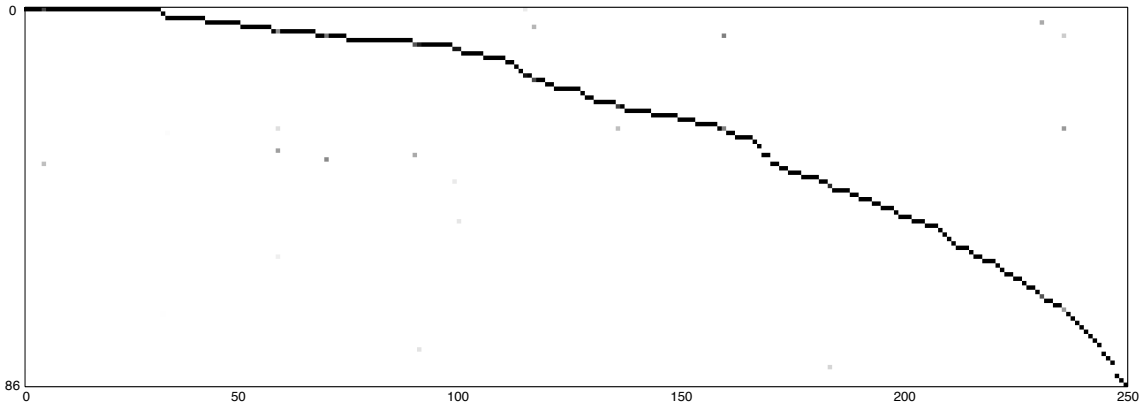
Next, consider the blue line, which corresponds to our improved instance-level overlap check (using ICP and volumetric overlap). Note the greatly improved precision with only a small cost in intra-instance recall. As before, inter-instance recall is zero.

Finally, consider the red lines, which include the full class-level association (including the instancewise edges). The recall values starts surprisingly high, and degrades slowly; the rightmost point (corresponding to training on all of the data) has a recall of 71.8%. By contrast, the precision grows quickly, surpassing convex hulls after training on three-fifths of the data, and topping out at 98.7%.

Our emphasis on high precision means that our clustering algorithm will likely split ground truth classes into several clusters. This *dispersion* is detailed in Figure 6.7.

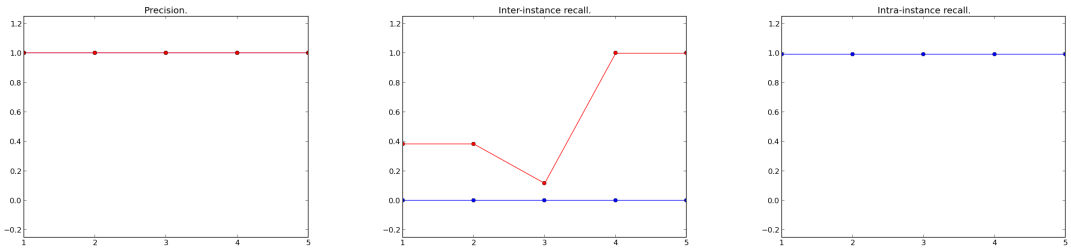


(a) Rows normalized to sum to 1. ("Dispersion")



(b) Columns normalized to sum to 1. ("Purity")

FIGURE 6.7: Cluster dispersion in the Willow Garage dataset. In the images above, each row corresponds to a ground-truth cluster, and each column to an inferred cluster (of which there are 250 that include at least one "good" segment). The rows are sorted by cluster size, with the largest cluster at the top. For easier visualization, the color scheme is reversed: black corresponds to 1, while white corresponds to 0. The pixel at row r and column c is the number of segments shared by ground-truth cluster r and inferred cluster c. In (a), the rows are normalized to sum to 1, meaning that each pixel represents what percentage of the ground-truth class is covered by each inferred cluster. As an example, consider the leftmost column. It contains two dark pixels; one at row 0, indicating that inferred cluster 0 contains nearly all of the elements of ground-truth cluster 0, and one near the bottom, indicating that it also contains all of the elements of some other class. (In this case, the other class has exactly one element.) The large horizontal gap between the upper-leftmost pixel and the first pixel in the second row indicates that ground-truth cluster 0 is broken into several inferred clusters; however, they are very small compared to cluster 0, which contains the great majority of the class. In (b), the *columns* are normalized to sum to 1, meaning that each value represents the percentage of that *inferred cluster* that belongs to a given ground-truth cluster. The very dark values are expected: they imply that each inferred cluster corresponds to very few different ground-truth clusters, which is to be expected given our emphasis on precision over recall.



(a) (b) (c)

FIGURE 6.8: Results of the deterministic clustering algorithm on the large run from the Mobile Objects dataset. The format of this figure follows Figure 6.6. Note the interesting "dip" in (b). In every example shown here, including three-fifths, training achieved 100% precision and 100% inter-instance recall on the training set. We hypothesize that the dip is due to there being more than one setting of the parameters that achieves this performance, and that we were simply unlucky in our choice. We achieve 100% precision and 100% inter-instance recall after training on four-fifths of the data, and the best parameters do not change when we train on all of the data.

Large Run

We also evaluated the nonprobabilistic approach on the large run from the Mobile Objects dataset (after applying stationary objects segmentation). As this run contains large, well-textured, well-separated objects, it is in almost every way easier than the Willow Garage dataset. We present our results in Figure 6.8 for completeness, and to confirm that our technique also applies to more-textured objects. The best parameters were $H = 0.35$, $V = 0.71$, and $F = 291$, which achieved both 100% precision and 100% inter-instance recall.

7

Conclusions & Future Work

In Chapter 1, we described a "home robot": one that operates in your environment and does your bidding. We motivated our work by touching on the many useful things such a robot could do, and by noting that they require the robot to understand objects.

Such a robot is not yet for sale, but think forward a few years, suppose that you have just bought one, and consider what happens when you open the box and turn it on for the first time. Your robot is *tabula rasa*, or nearly so: it knows nothing about you, your house, or your objects. If your robot is anything like a PR2, the very first step is calibrate the robot, and the second is SLAM (to support autonomous navigation). After that, your robot will need to know something about your objects. Before object discovery, this was your responsibility: you had to show the robot each object many times. That process is both labor-intensive and error-prone: you are in for many hours of tedious work.

Our algorithms replace that step. With our algorithms running, the robot will *teach itself* about your objects, and will only need human input when a new class has been conclusively discovered. That question is of the form "I have discovered a

new class; what is it called?" which is far easier (and rarer) than manually labeling images.

We do not require that you live in a robotics lab for this to work. We evaluated our work in a realistic setting by collecting a dataset of 67 runs through an office environment. On this dataset, which is the largest publicly available, we segment objects from their background with 85.6% precision, and then associate them into classes with 98.7% precision and 71.8% recall (meaning that we recovered nearly three-quarters of the inter-class relationships with nearly no mistakes).

With our public code release, our algorithms can be deployed on robots today. As our data are public as well, they can form a benchmark against which future work in object discovery can measure itself.

Object discovery also presents a variety of possibilities for future work, both in improved discovery algorithms and in applications of discovery to higher-level robotics problems. We present a few examples here.

7.1 Supervised Training

This work has focused almost exclusively on unsupervised techniques, and left aside supervised object recognition. However, given sufficient training data, supervised algorithms have shown excellent performance. By leveraging supervised recognition, we could likely improve our segmentation and association results.

The idea is this: take a cluster produced by one of our unsupervised algorithms, and feed it as input into the training stage of a supervised recognition algorithm. After training, the new recognizer would be added to the system, to be run in parallel with other discovery or recognition algorithms.

I have done some (very preliminary) work in this area, using the tree of deformable parts model described by Yang and Ramanan (2011) and Felzenszwalb et al. (2010). With a short, close-range, high-resolution dataset, containing a small

corpus of objects, it works: we can successfully train tree-of-parts models that discover new inter-instance associations. However, when applied to the Willow Garage dataset, this technique fails. We hypothesize that this failure (like that of visual features on the same dataset) is due to the nature of our data. The tree-of-parts model uses a histogram of oriented gradients (or "HOG", a subset of SIFT) features for its parts, and many of our segments are simply too small (or too featureless) to contain meaningful gradients. As an example, consider the houseplants in Figure 1.3; these segments contain almost no visual structure. Higher-resolution data, active search strategies (discussed below) or simply other supervised algorithms (perhaps leveraging depth) could fix this. All of these are promising directions for future work.

7.2 Active Search

Another possibility for improved discovery is *active search*: searching out objects to discover, rather than finding them "by accident". I see two natural directions for future work in discovery with active search. At a low level, each supporting surface could be actively scanned. Instead of driving past a surface and observing it incidentally, the robot could plan a sequence of observations that observe every part of the surface from multiple points of view. In the extreme case, a technique like KinectFusion (Newcombe et al., 2011; Izadi et al., 2011) or that of Karpathy et al. (2013) could be used to build an extremely high-resolution three-dimensional model of the entire tabletop. The resulting models would likely permit extremely accurate segmentation.

At a high level, a robot could seek out the surfaces using an area-sweeping strategy, or by checking on surfaces that were previously observed. Combining these high- and low-level improvements would yield an extremely thorough search algorithm, and a deployed robot could trade off between active and passive search as resources and requirements permit.

7.3 Object Patterns

Consider the state of our system after being run on the Willow Garage dataset: many objects from many classes, associated across time and space. Furthermore, assume that these classes have been manually labeled, so that the robot can understand a query like "Robot, bring me a coffee cup." The robot is now faced with a search problem: where should it look? If a coffee cup was seen very recently, and nearby, the robot can just navigate there. Otherwise, it must plan a search strategy.

The input to such a planner is a set of object "observation events", (that is, labeled segments) each with a three-dimensional location and a timestamp. Possible approaches range from simple spatial clustering (which would find groups of events) to information-theoretic search strategies that seek to minimize the uncertainty about object locations.

In Section 6.1, we described a generative model in which the "world" generates "objects". This model ignores location: sampling from the "world" occurs without regard to the object's location. Clearly, this ignores information: coffee cups happen most often in some settings, computer monitors in others, and so on. A more detailed model could include a spatial component, so that *locations* generate "objects". Given a particular object class, the distribution over where that object appears could be used to plan a better search strategy. Similarly, the location of an observation could be included when calculating the likelihood that a given segment was generated by a given class, which could improve clustering accuracy. The details of such a model, and of how to use it, present a particularly interesting direction for future work.

7.4 Long-term Deployment

The Willow Garage dataset is already the longest dataset of its type, but its duration was limited by human time, not computational or robotic resources. We have

demonstrated object discovery over a period of six weeks, but our system scales to (at least) months, and deployments of this duration should be attempted.

A particularly interesting question is to ask how the quality of object discovery evolves over time, in terms of the number of discovered instances and classes, and in terms of the quality of inter-instance association. This could allow a user to predict how long it would take a robot to learn a representative set of the objects in their environment. Another experiment that should be run is to roll the system out in a different indoor environment: while our techniques (and parameters) show good generalization over our data, we still run the risk of having overfit to Willow Garage itself. A new environment would allow us to investigate this possibility.

Bibliography

Anguelov, D., Biswas, R., Koller, D., Limketkai, B., and Thrun, S. (2002), "Learning Hierarchical Object Maps of Non-Stationary Environments With Mobile Robots," in *Conference on Uncertainty in Artificial Intelligence*, pp. 10–17.

Ayvaci, A. and Soatto, S. (2012), "Detachable Object Detection: Segmentation and Depth Ordering from Short-Baseline Video," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34, 1942–1951.

Bersch, C., Pangercic, D., Osentoski, S., Hausman, K., Marton, Z.-C., Ueda, R., Okada, K., and Beetz, M. (2012), "Segmentation of Textured and Textureless Objects through Interactive Perception," in *RSS Workshop on Robots in Clutter: Manipulation, Perception and Navigation in Human Environments*, Sydney, Australia.

Besl, P. J. and McKay, N. D. (1992), "A Method for Registration of 3-D Shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14, 239–256.

Biswas, R., Limketkai, B., Sanner, S., and Thrun, S. (2002), "Towards Object Mapping in Non-Stationary Environments With Mobile Robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1014–1019.

Blodow, N., Jain, D., Marton, Z., and Beetz, M. (2010), "Perception and Probabilistic Anchoring for Dynamic World State Logging," in *IEEE-RAS International Conference on Humanoid Robots*, pp. 160–166.

Costeira, J. P. and Kanade, T. (1998), "A Multibody Factorization Method for Independently Moving Objects," *International Journal of Computer Vision*, 29, 159–179.

Dellaert, F., Fox, D., Burgard, W., and Thrun, S. (1999), "Monte Carlo Localization for Mobile Robots," in *IEEE International Conference on Robotics and Automation*, pp. 1322–1328.

Deyle, T., Tralie, C. J., Reynolds, M. S., and Kemp, C. C. (2013), "In-Hand Radio Frequency Identification (RFID) for Robotic Manipulation," in *IEEE International Conference on Robotics and Automation*.

Edelsbrunner, H. and Mücke, E. P. (1994), "Three-dimensional Alpha Shapes," *ACM Transactions on Graphics*, 13, 43–72.

Eliazar, A. (2005), "DP-SLAM," Ph.D. thesis, Duke University.

Eliazar, A. and Parr, R. (2003), "DP-SLAM: Fast, Robust Simultaneous Localization and Mapping Without Predetermined Landmarks," in *International Joint Conference on Artificial Intelligence*, vol. 18, pp. 1135–1142, Citeseer.

Eliazar, A. and Parr, R. (2004), "DP-SLAM 2.0," in *IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1314–1320.

Eliazar, A. and Parr, R. (2005), "Hierarchical Linear/Constant Time SLAM Using Particle Filters for Dense Maps," *Advances in Neural Information Processing Systems*, 18.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2010), "The Pascal Visual Object Classes (VOC) Challenge," *International Journal of Computer Vision*, 88, 303–338.

Fairfield, N. (2009), "Localization, Mapping, and Planning in 3D Environments," Ph.D. thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Felzenszwalb, P., Girshick, R., McAllester, D., and Ramanan, D. (2010), "Object Detection with Discriminatively Trained Part Based Models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32, 1627–1645.

Felzenszwalb, P. F. and Huttenlocher, D. P. (2004), "Efficient Graph-Based Image Segmentation," *International Journal of Computer Vision*, 59, 167–181.

Ferguson, T. S. (1973), "A Bayesian Analysis of Some Nonparametric Problems," *Annals of Statistics*, 1, 209–230.

Fischler, M. A. and Bolles, R. C. (1980), "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications of the ACM*, 24, 381–395.

Fox, D. (2003), "Adapting the Sample Size in Particle Filters Through KLD-Sampling," *International Journal of Robotics Research*, 22, 985–1003.

Galindo, C., Saffiotti, A., Coradeschi, S., Buschka, P., Fernandez-Madrigal, J., and Gonzalez, J. (2005), "Multi-Hierarchical Semantic Maps for Mobile Robotics," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2278–2283.

Galindo, C., Fernández-Madrigal, J.-A., González, J., and Saffioti, A. (2008), "Robot Task Planning Using Semantic Maps," *Robotics and Autonomous Systems*, 56, 955–966.

Gibson, J. J. (1986), *The Ecological Approach to Visual Perception*, Psychology Press.

Grisetti, G., Stachniss, C., and Burgard, W. (2005), "Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling," in *IEEE International Conference on Robotics and Automation*, pp. 2432–2437.

Grisetti, G., Tipaldi, G., Stachniss, C., Burgard, W., and Nardi, D. (2007a), "Fast and Accurate SLAM with Rao-Blackwellized Particle Filters," *Robotics and Autonomous Systems*, 55, 30–38.

Grisetti, G., Stachniss, C., and Burgard, W. (2007b), "Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters," *IEEE Transactions on Robotics*, 23, 34–46.

Henry, P., Krainin, M., Herbst, E., Ren, X., and Fox, D. (2010), "RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments," in *International Symposium on Experimental Robotics*, vol. 20, pp. 22–25.

Herbst, E., Ren, X., and Fox, D. (2011a), "RGB-D Object Discovery via Multi-Scene Analysis," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1–7.

Herbst, E., Henry, P., Ren, X., and Fox, D. (2011b), "Toward Object Discovery and Modeling via 3-D Scene Comparison," in *IEEE International Conference on Robotics and Automation*, pp. 2623–2629.

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W. (2013), "OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees," *Autonomous Robots*, Software available at `http://octomap.github.com`.

Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R. A., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A., and Fitzgibbon, A. (2011), "KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera," in *ACM Symposium on User Interface Software and Technology*.

Janoch, A., Karayev, S., Jia, Y., Barron, J. T., Fritz, M., Saenko, K., and Darrell, T. (2011), "A Category-Level 3D Object Dataset: Putting the Kinect to Work," in *International Conference on Computer Vision Workshop on Consumer Depth Cameras in Computer Vision*.

Kang, H., Efros, A. A., Hebert, M., and Kanade, T. (2009), "Image Composition for Object Pop-out," in *International Conference on Computer Vision Workshop on 3D Representation for Recognition*.

Kang, H., Hebert, M., and Kanade, T. (2011), "Discovering Object Instances from Scenes of Daily Living," in *IEEE International Conference on Computer Vision*, pp. 762–769.

Kang, H., Hebert, M., Efros, A. A., and Kanade, T. (2012), "Connecting Missing Links: Object Discovery from Sparse Observations using 5 Million Product Images," in *European Conference on Computer Vision*, pp. 794–807.

Karpathy, A., Miller, S., and Fei-Fei, L. (2013), "Object Discovery in 3D Scenes via Shape Analysis," in *IEEE International Conference on Robotics and Automation*.

Khoshelham, K. and Elberink, S. O. (2012), "Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications," *Sensors*, 12, 1437–1454.

Konolige, K. (2010), "Projected Texture Stereo," in *IEEE International Conference on Robotics and Automation*, pp. 148–155.

Krainin, M., Henry, P., Ren, X., and Fox, D. (2011), "Manipulator and object tracking for in-hand 3D object modeling," *The International Journal of Robotics Research*, 30, 1311–1327.

Lai, K., Bo, L., Ren, X., and Fox, D. (2011), "A Large-Scale Hierarchical Multi-View RGB-D Object Dataset," in *IEEE International Conference on Robotics and Automation*.

Lampert, C. H., Blaschko, M. B., and Hofmann, T. (2009), "Efficient Subwindow Search: A Branch and Bound Framework for Object Localization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31, 2129–2142.

Lowe, D. (1999), "Object recognition from local scale-invariant features," in *International Conference on Computer Vision*, vol. 2, pp. 1150 –1157.

Marder-Eppstein, E., Berger, E., Foote, T., Gerkey, B., and Konolige, K. (2010), "The Office Marathon: Robust navigation in an indoor office environment," in *IEEE International Conference on Robotics and Automation*, pp. 300–307.

Mason, J. and Marthi, B. (2012), "An Object-Based Semantic World Model for Long-Term Change Detection and Semantic Querying," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3851–3858.

Mason, J., Ricco, S., and Parr, R. (2011), "Textured Occupancy Grids for Monocular Localization Without Features," in *IEEE International Conference on Robotics and Automation*.

Mason, J., Marthi, B., and Parr, R. (2012), "Object Disappearance for Object Discovery," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2836–2843.

Mason, J. M. (2010), "Stereo Mapping for Textured 3D Occupancy," My preliminary exam document.

Meagher, D. (1982), "Geometric Modeling Using Octree Encoding," *Computer Graphics and Image Processing*, 19, 129–147.

Modayil, J. and Kuipers, B. (2004), "Towards Bootstrap Learning for Object Discovery," in *AAAI Workshop on Anchoring Symbols to Sensor Data*.

Moravec, H. (2002), "Robust Navigation by Probabilistic Volumetric Sensing," Tech. rep., Carnegie Mellon University, `http://www.frc.ri.cmu.edu/~hpm/project.archive/robot.papers/2002/ARPA.MARS/Report.0202.html`.

Moravec, H. and Elfes, A. E. (1985), "High Resolution Maps from Wide Angle Sonar," in *IEEE International Conference on Robotics and Automation*, pp. 116 – 121.

Neal, R. M. (2000), "Markov Chain Sampling Methods for Dirichlet Process Mixture Models," *Journal of Computational and Graphical Statistics*, pp. 249–265.

Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohli, P., Shotton, J., Hodges, S., and Fitzgibbon, A. (2011), "KinectFusion: Real-time dense surface mapping and tracking," in *IEEE International Symposium on Mixed and Augmented Reality*, pp. 127–136.

Ng, A. Y., Gould, S., Quigley, M., Saxena, A., and Berger, E. (2007), "STAIR: Hardware and Software Architecture," in *AAAI Robotics Workshop*.

Ng, A. Y., Gould, S., Quigley, M., Saxena, A., and Berger, E. (2008), "STAIR: The STanford Artificial Intelligence Robot projec," in *Snowbird*.

Nüchter, A., Wulf, O., Lingemann, K., Hertzberg, J., Wagner, B., and Surmann, H. (2006), "3D Mapping with Semantic Knowledge," in *Robocup International Symposium*, eds. A. Bredenfeld, A. Jacoff, I. Noda, and Y. Takahashi, pp. 335–346, Springer-Verlag.

Pfister, H., Zwicker, M., van Baar, J., and Gross, M. (2000), "Surfels: Surface Elements as Rendering Primitives," in *Conference on Computer Graphics and Interactive Techniques*, pp. 335–342.

Pirker, K., Rüther, M., and Bischof, H. (2010), "Histogram of Oriented Cameras — a New Descriptor for Visual Slam in Dynamic Environments," in *British Machine Vision Conference*.

Pronobis, A. (2011), "Semantic Mapping with Mobile Robots," Ph.D. thesis, KTH.

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., and Ng, A. Y. (2009), "ROS: an open-source Robot Operating System," in *Proceedings of the ICRA workshop on Open-Source Software*.

Ranganathan, A. and Dellaert, F. (2007), "Semantic Modeling of Places using Objects," in *Robotics Science and Systems*.

Rao, S. R., Tron, R., Vidal, R., and Ma, Y. (2008), "Motion Segmentation via Robust Subspace Separation in the Presence of Outlying, Incomplete, or Corrupted Trajectories," in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8.

Rother, C., Minka, T., Blake, A., and Kolmogorov, V. (2006), "Cosegmentation of Image Pairs by Histogram Matching — Incorporating a Global Constraint into MRFs," in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. 993–1000.

Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011), "ORB: an Efficient Alternative to SIFT or SURF," in *International Conference on Computer Vision*, pp. 1–8.

Rusu, R. B. and Cousins, S. (2011), "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation*, pp. 1–4.

Rusu, R. B., Blodow, N., Marton, Z. C., and Beetz, M. (2009a), "Close-range Scene Segmentation and Reconstruction of 3D Point Cloud Maps for Mobile Manipulation in Domestic Environments," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1–6.

Rusu, R. B., Holzbach, A., Beetz, M., and Bradski, G. (2009b), "Detecting and Segmenting Objects for Mobile Manipulation," in *IEEE International Conference on Computer Vision Workshops*, pp. 47–54.

Rusu, R. B., Blodow, N., and Beetz, M. (2009c), "Fast Point Feature Histograms (FPFH) for 3D Registration," in *IEEE International Conference on Robotics and Automation*, pp. 3212–3217.

Rusu, R. B., Bradski, G., Thibaux, R., and Hsu, J. (2010), "Fast 3D Recognition and Pose Using the Viewpoint Feature Histogram," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2155–2162.

Sanders, B. C. S., Nelson, R. C., and Sukthankar, R. (2002), "A Theory of the Quasi-static World," in *International Conference on Pattern Recognition*, vol. 3, pp. 1–6.

Sivic, J. and Zisserman, A. (2006), "Video Google: Efficient Visual Search of Videos," in *Toward Category-Level Object Recognition*, eds. J. Ponce, M. Hebert, C. Schmid, and A. Zisserman, vol. 4170, pp. 127–144, Springer.

Sivic, J. and Zisserman, A. (2008), "Efficient Visual Search for Objects in Videos," *Proceedings of the IEEE*, 96, 548–566.

Sivic, J., Schaffalitzky, F., and Zisserman, A. (2004), "Efficient Object Retrieval from Videos," in *European Signal Processing Conference*.

Southey, T. and Little, J. J. (2006), "Object Discovery through Motion, Appearance and Shape," *AAAI Workshop on Cognitive Robotics*.

Stachniss, C., Hähnel, D., Burgard, W., and Grisetti, G. (2005), "On Actively Closing Loops in Grid-based FastSLAM," *Advanced Robotics*, 19, 1059–1080.

Szeliski, R. (2010), *Computer Vision: Algorithms and Applications*, Springer.

Taylor, G. (2011), "Feature Selection for Value Function Approximation," Ph.D. thesis, Duke University.

Thrun, S., Burgard, W., and Fox, D. (2005), *Probabilistic Robotics*, MIT Press Cambridge, MA, USA.

Tomasi, C. and Kanade, T. (1992), "Shape and Motion from Image Streams under Orthography: a Factorization Method," *International Journal of Computer Vision*, 9, 137–154.

Trevor, A. (2012), "PCL::Segmentation — planes, clusters, and more," In PCL tutorial at IROS 2012. Currently unpublished material, but available at pointclouds.org.

Vasudevan, S., Gachter, S., Nguyen, V., and Siegwart, R. (2007), "Cognitive maps for mobile robots — an object based approach," *Robotics and Autonomous Systems*, 55, 359–371.

Vicente, S., Kolmogorov, V., and Rother, C. (2010), "Cosegmentation Revisited: Models and Optimization," in *European Conference on Computer Vision*, pp. 465–479.

Yang, Y. and Ramanan, D. (2011), "Articulated pose estimation with flexible mixtures-of-parts," in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1385–1392.

Zender, H., Martínez Mozos, O., Jensfelt, P., Kruijff, G.-J., and Burgard, W. (2008), "Conceptual Spatial Representations for Indoor Mobile Robots," *Robotics and Autonomous Systems*, 56, 493–502.

Biography

Julian Mac Neille Mason was born on April 14th, 1984, in Anchorage, Alaska. (For the curious, that's three names: "Julian", "Mac Neille", and "Mason", and yes, there's a space in one of them.) Because of the sheer number of Julians in his family, everybody has always called him Mac. He grew up in Anchorage, graduating from the SWS program at East Anchorage High School in 2002. After that, he attended Harvey Mudd College in Claremont, California, graduating with a B.S. in Computer Science in 2006. He started in the Ph.D. program at Duke University immediately afterward, and successfully defended his Ph.D. in Computer Science (with a Certificate in College Teaching) in June of 2013.

His published papers include "Textured Occupancy Grids for Monocular Localization Without Features" (Mason et al., 2011), "An Object-Based Semantic World Model for Long-Term Change Detection and Semantic Querying" (Mason and Marthi, 2012), and "Object Disappearance for Object Discovery" (Mason et al., 2012). He has also released the software developed for those papers, and the datasets on which they were evaluated (see the papers themselves, or this document, for details).

He will start at Google Research in the fall of 2013.